A CATEGORY THEORETIC APPROACH TO PLANNING IN A COMPLEX WORLD

A prospectus submitted in partial fulfillment of the degree of Doctor of Philosophy

Preliminary Oral Examination for
ANGELINE AGUINALDO

Advisor:
WILLIAM REGLI

Committee Members:
JOHN YIANNIS ALOIMONOS
DANA NAU
MARK FUGE

Department of Computer Science
University of Maryland, College Park, MD 20742
May 23, 2023

Abstract

Artificial intelligence (AI) planning has become a crucial component in the development of intelligent systems, facilitating the generation and adaptation of plans to achieve specific goals in complex environments. This thesis addresses the challenge of task planning in complex domains, especially ones with a large number of objects and properties, rich relations and dependencies, and incomplete or evolving knowledge. Our approach aims to tackle three specific problems: the difficulty in expressing complex and structured world states, the challenges in reasoning about transitive relations, and the integration of AI components for neurosymbolic reasoning. To address these issues, we propose a novel language called AlgebraicPlanning, based on C-sets and DPO rewriting from category theory. This language uses the expressive power of ontology-based typed graphs to capture complex world states and the relationships within them. We hypothesize that our planning framework will yield higher quality plans and enhance the applicability of task planning in complex real-world domains relative to existing planners. To validate this, we will design a method for assessing the plan length, stability, success rate, and efficiency of plans generated by our framework compared to other planning algorithms. We will also explore the application of our approach in diverse domains, such as classical planning domains, manufacturing, and home service robotics. The contributions of this thesis are expected to include (i) a robust language for representing complex world states, (ii) a planning algorithm based on this formalism, (iii) a method for plan quality assessment, (iv) diverse case studies, and an (v) application of category theory in AI planning. These advancements will pave the way for more robust and adaptive intelligent systems capable of navigating and executing tasks in complex environments.

Contents

List of Figures

Chapter 1

Introduction

Over the past few decades, artificial intelligence (AI) planning has grown to be a critical component in the development of intelligent systems, enabling them to reason about actions and achieve specific goals within complex environments. A complex environment refers to a scenario or setting with multiple factors, conditions, or elements that make the planning and execution of tasks more challenging. These factors can be dynamic, uncertain, or interconnected, and they can influence the robot's decision-making process. From the viewpoint of symbolic task planning, a complex world state refers to a situation where the representation of the environment and the relationships between various elements involve multiple factors that make the planning problem more challenging. These factors can be related to the number of objects, properties, relations, and constraints in the environment, as well as the intricacies of the agent's goals and the interactions between different agents [37, 71, 32]. In this thesis, we would like to focus particular attention on world states that have large number of objects and properties, rich relations and dependencies, and incomplete or changing knowledge. With this in mind, we will not consider the issues related to hierarchical goals, temporal constraints, and multi-agent scenarios. This restricted view of world state complexity is useful in scenarios where a robot needs to perform a task in environments that have many object and object interactions in the world, such as in an assembly line scenario in manufacturing.

For example, imagine a task planning scenario in a large factory with a highly automated assembly line for producing electric vehicles like one shown in Figure 1.1. The factory floor contains a diverse range of machines, robots, and human operators working together in a coordinated manner. The assembly line is composed of different stations, each responsible for a specific part of the vehicle assembly process. In this complex environment, the world model would need to represent various objects and relations, such as different types of machines, like welding machines, painting machines, and assembly machines; various vehicle parts, such as, frames, engines, wheels, batteries, electronic components, and interior and exterior parts; various compatibility between machines, robots, and vehicle parts; spatial relations (e.g., locations of machines, robots, and workstations); and temporal relations (e.g., scheduling and precedence constraints). Now, although, various types of robots, such as mobile robots for transportation, or robotic arms for manipulation and assembly, interact with the different workstations, and therefore require different task plans– all robots on the factory floor would benefit from a task planner that (a) was able to reason over a similarly large and complex world model and (b) was reasonably stable against small changes in the world.

Defining a formalism that is capable of managing many features of the world within task planning is among the major contributions of this thesis. As mentioned, there are a number of considerations when designing a framework for task planning over complex domains. A summary of the three specific problems we aim to tackle in this thesis are provided below.

Figure 1.1: An assembly line equipped with many specialized robots

1. **Difficulty in expressing complex and structured world states.** Current representation frameworks struggle to capture the rich relations and intricate structure of real-world environments, limiting the expressiveness of planning domains [37, 86]. This makes it difficult to leverage domain-specific knowledge and accurately model complex world states with numerous objects and relationships. Although, object typing is included in PDDL 1.2 [51], there is limited consideration for sophisticated type hierarchies and relations within planning domains in practice. For instance, in the Planners page of the Planning.wiki, only seven [62, 9, 31, 5, 38, 43, 42] out of 107 planners mention the use of either at type system or the `:typing` extension in their work. In these cases typing is used for validating predicates at best, and not for making more sophisticated inferences based on the world models. Additionally, existing planning methods do not scale well for a large number of literals– when the number of literals in the planning problem grows, the complexity of the planning problem increases [37]. This can make planning intractable for large and complex world states, especially those derived from typed graphs like scene graphs.

2. **Reasoning about transitive relations (composition)** More specifically, current methods do not consider implicit transitive dependencies formed by relations. As a result, they may struggle to reason about indirect dependencies and interactions that arise from these relationships. In this thesis, we are particularly concerned with the language's ability to reason about transitive closure, which we term composition (of relations or predicates) as we believe this feature will improve the ability to generate high-quality plans for complex domains. Currently, transitive relations need to be represented through a thorough and precise enumeration of the relevant propositions or through domain axioms [80]. Both of which increase the complexity of the search space. In the case of domain axioms, more testing is required for ensure its stability as a syntax component [30].

3. **Difficulty integrating with other AI components to support neurosymbolic reasoning.** Integrating task planning with other essential AI components, such as perception and control, remains a challenge [77]. This is largely due to representation mismatch. Task planners use symbolic, logic-

based languages, while neural approaches for perception use tensors for which both semantics need to be defined. Here lies a need for an interoperable language that bridges the gap between both representations. Integrating these representations can be challenging because it often requires finding an adequate abstraction for continuous and comprehensive features of the world [3, 50, 34]. Advancements toward perceptual formalisms like scene graphs provide opportunities for integrating neural and symbolic knowledge. We see our approach as a way of tackling the problem of integrating neural with symbolic planning by proposing a planning representation that acts on typed graphs while preserving relationships accounted for by the ontology.

We hope that by addressing these open problems using our planning framework, we will be able to generate higher quality plans for a range of complex domains.

1.1 Research Questions

Throughout this thesis, we will focus on four guiding research questions.

RQ1 How does the use of more descriptive and structured formalisms for world states during planning, such scene graphs and ontologies, broaden the applicability of task planning to a wider range of complex real-world domains?

RQ2 What modifications to existing ontology-based planning architectures can be made in order to preserve the ontological structure throughout the planning process?

RQ3 Can category theory improve the generation of high quality plans for complex domains by providing clear and consistent formal semantics for typing and transitive closure for task planning?

RQ4 To what extent does the development of a planning language with clear and consistent formal semantics for typing and transitive closure improve the quality of planning solutions, especially for complex domains?

1.2 Our approach

Complex world states, which often involve numerous objects and intricate relationships, necessitate a robust representation language capable of capturing the intricacies of the domain. In this thesis, we propose a novel language, called AlgebraicPlanning, based on C-sets and DPO rewriting from category theory that leverages the expressive power of ontology-based typed graphs to manage complex world states and the relationships between objects within them. We assume a fully observable and deterministic environment. Common formalisms in knowledge representation and robotics, such as knowledge graphs and scene graphs, are expressible as typed graphs, making this a useful language for reasoning about task plans for robotics applications.

Additionally, to measure the extent to which our planning framework improves the quality of planning solutions, we will design a method for assessing the length of plan, success rate, and efficiency in terms of time and memory between our planning framework and other planning algorithms. The most relevant planners are ones that handle typing and transitive closure, namely Metric-FF [42] with and without derived axioms for transitive closure [80]. We also introduce a method for assessing stability [29] of plans relative

to changes in the planning problem. A prequisite to this method is a novel measurement that we developed for quickly assessing the distance between planning problems to ensure a fair distribution of problems in simulation. By providing a systematic way to examine plan stability per changes in the world, we also gain insight into the relevance of features in the world to achieving a goal.

Through the development of a rich representation language and a method for assessing plan quality, this thesis aims to contribute to the advancement of AI task planning in dynamic environments, paving the way for more robust and adaptive intelligent systems. In summary, the following hypothesis is brought forth:

Hypothesis A. We hypothesize that by developing a planning framework that effectively addresses the challenges of:

(i) representing complex and structured world states,

(ii) reasoning about transitive relations, and

(iii) remaining interoperable with other AI components for neurosymbolic reasoning,

we will be able to generate higher quality plans and enhance the applicability of task planning in complex real-world domains relative to existing planners.

For this work, we make the following assumptions:

Assumption I. Scene graphs are expressive enough for representing complex and structured world states for task planning [54, 6, 21].

Assumption II. All typed graphs can be represented as C-sets [18].

Given these assumptions, the main hypothesis can be broken down into the following:

Hypothesis A.1 In support of (i), we claim that all scene graphs can be represented as typed graphs.

Hypothesis A.2 In support of (ii), we claim that C-sets and DPO rewriting preserves transitive relations expressed in the ontology after every action.

Hypothesis A.3 In support of (iii), we claim that arbitrary scene graphs based on a large number of entities and relations can be translated into C-sets in polynomial time.

Hypothesis A.4 In support of higher quality plans, we claim that the proposed formalism produces more concise plans (fewer steps) at an equal or higher success rate than an existing general-purpose planner, Metric-FF [42], that considers typing with and without transitive closure domain axioms for a range of domains (those with at least k-entities and m-relations in their ontology, where $k = [5, 10, 100]$, for now, and m will be determined) and problem sizes (those with n-objects in the domain, where $n = [5 : 1000]$ will depend on the specific domain).

Hypothesis A.5 In support of higher quality plans, we claim that the proposed formalism produces more stable plans than an existing general-purpose planner, Metric-FF [42], that considers typing with and without transitive closure domain axioms for a range of domains (those with at least k-entities and

m-relations in their ontology, where $k = [5, 10, 100]$, for now, and m will be determined) and problem sizes (those with n-objects in the domain, where $n = [5 : 1000]$ will depend on the specific domain).

So far, we have defined the mathematical formalism, developed a prototype forward planning algorithm, and begun developing the infrastructure for testing. We aim to definitively evaluate and refine each hypothesis during the remainder of this work.

1.3 Contributions

We expect this work to have the following contributions:

1. **CONTRIBUTION 1 A rich representation planning language based on category theory:** Design and implement a robust language for representing complex world states that can capture various types of world changes. The language should be expressive enough to model intricate relationships between objects and facilitate efficient reasoning about the impact of world changes on plan stability.
2. **CONTRIBUTION 2 A planning algorithm based on this formalism:** Identify a performant state-based planning algorithm and adapt it to work with the proposed representation language.
3. **CONTRIBUTION 3 A method for plan quality assessment:** Develop a method for assessing the quality of plans, including their stability to changes in the world state. This method should provide a measurement for assessing differences in planning problem to ensure diverse planning problems are properly represented.
4. **CONTRIBUTION 4 Case studies exploring planning in diverse domains:** Investigate the applicability of the proposed approach in different domains, such as classical planning domains, manufacturing, and home service robotics, where complex world states and dynamic environments can be studied. This exploration can help identify potential improvements and extensions to our approach and demonstrate its relevance in various application areas.
5. **CONTRIBUTION 5 An example and application based wholly in category theoretic formalisms:** Demonstrate the application of category theoretic formalisms to AI task planning at a variety of problem and domain scales. This exploration provides concrete examples towards applications of category theory.

Proposal Structure

This thesis proposal is organized into seven chapters that provide a comprehensive exposition of the proposed approach. **Chapter 1** provides the motivation and problem definition for the research, outlines the research objectives, and presents an overview of the thesis structure. **Chapter 2** reviews background and related work in knowledge representation for planning, task planning, and category theory. **Chapter 3** discusses preliminary results using the proposed formalism to model complex world states. **Chapter 4** discusses preliminary results using the proposed formalism for planning. **Chapter 5** provides a sketch of ongoing results related to measuring problem difference. **Chapter 6** summarizes the proposed work, evaluation method, and timeline. **Chapter 7** concludes the proposal by summarizing the problem and proposed contributions. **Appendix A** provides a examples of PDDL domain and problem files for the anticipated experimental domains. **Appendix B** provides copies of the AlgebraicPlanning implementation. **Appendix**

C provides a list of papers, presentations, and extracurricular contributions. **Appendix D** provides the reading list for all three research areas.

Chapter 2

Background and Related Work

This chapter is broken out into three main sections: Section 2.1 Knowledge Representation for Planning, Section 2.3 Task planning, and Section 2.4 Category Theory. These are the main topic areas that will be leveraged in this work. Here we will provide a background and summary of related work.

2.1 Knowledge Representation for Planning

In this section, we will summarize background and relevant work towards integrating semantic information using ontology and scene graph formalisms in the task planning process. We aim to examine how semantic information could be leveraged in task planning to improve its range of applicability to complex real-world domains.

2.1.1 Existing Methods

There are few candidates for world state representations that capture the desired semantics in a formal language. The two candidates we compare in this thesis are typed propositions and typed graph formalisms for world states for their ability to handle typed objects and relations.

2.1.1.1 Using typed propositions

The STRIPS-based representation of world states is based on a restricted form of first-order logic in which the world states are represented as a conjunction of literals, where a literal is an atomic proposition or its negation [37]. These propositions can encode facts such as the location of objects, the state of various sensors, and other relevant information. Logical operators such as AND, OR, and NOT are used to combine these propositions into more complex expressions that can be used to describe the relationships between different parts of the world state. World states can be lifted to abstract states consisting of a conjunction of predicates, where predicates are n-ary relations between typed variables. For example, the predicate `on(x - Block, y - Block)`might represent the fact that a block x is on top of another block y or, equally plausible, the converse. It is important to emphasize that the semantics of the predicates must be established informally, say through external documentation or word of mouth. A predicate becomes a grounded literal when the variable symbols, like x and y, are assigned to symbols that are constant. For instance, the grounded literal `on(b1 - Block, b2 - Block)` could represent a block called `b1` that is on top of a block called `b2`.

2.1.1.2 Using typed graphs

Typed graphs are powerful tools for adding structure to the objects and relations in robotics applications. A definition for typed graphs can be found in Definition 2.1.1 [46].

Definition 2.1.1. Let $T = (V_T, E_T)$ be a type graph, where V_T is a set of node types and E_T is a set of edge types. A typed graph $G = (V, E, \tau_v, \tau_e)$ consists of the following components:

- V is a set of nodes, and E is a set of edges, where each edge is an ordered pair of nodes $(u, v) \exists V \times V$.
- $\tau_v : V \to V_T$ is a node typing function that assigns a type from the set of node types V_T to each node in V.
- $\tau_e : E \to E_T$ is an edge typing function that assigns a type from the set of edge types E_T to each edge in E.

The typing functions τ_v and τ_e are used to associate types with nodes and edges in the graph. They ensure that each node and edge in the graph G has a corresponding type in the type graph T.

This provides a means to model and reason about the relationships between objects in the environment based on a type graph, or an ontology. Applications of this can be found amoung the work done to leverage knowledge graphs [77, 44, 11, 13] and scene graphs [21, 6] as a formalism for modeling world states in robotics.

Knowledge graphs Knowledge graphs are a form of knowledge representation that model relationships between entities using graph-based structures [44]. They consist of nodes, which represent entities such as objects, actions, or concepts, and edges, which represent relationships between these entities. Knowledge graphs provide a flexible and expressive way to represent structured information and enable efficient querying and reasoning capabilities. In robotics, knowledge graphs are employed to represent and reason about the relationships between various elements in the environment, such as: spatial relationships between objects and robots, e.g., "object A is on the left of object B"; temporal relationships between events and actions, e.g., "action A must be completed before action B"; semantic relationships between concepts and entities, e.g., "object A is a type of container." Knowledge graphs are particularly useful for high-level reasoning tasks in robotics, such as symbolic planning, where the robot must consider complex relationships and constraints to generate an effective plan.

Scene graphs Scene graphs are a specialized form of knowledge graphs used primarily in computer graphics and robotics to model the hierarchical structure of a scene [54, 21]. They consist of nodes representing objects, actions, and spatial relationships, arranged in a tree-like structure that reflects the hierarchical organization of the scene. Scene graphs enable efficient representation, manipulation, and rendering of complex scenes by organizing the information in a structured manner. In robotics, scene graphs are employed to represent the geometric and topological structure of the environment, such as: the hierarchical organization of objects and their parts, e.g., "object A is composed of parts B and C"; the spatial relationships between objects, e.g., "object A is inside object B"; the relative poses and transformations between objects, e.g., "object A is rotated 90 degrees with respect to object B" Scene graphs are particularly useful for geometric reasoning tasks in robotics, such as motion planning, navigation, and object manipulation, where the robot must consider the spatial structure of the environment to plan and execute its actions.

We deem typed graphs to be more favorable than typed predicates for the added structure provided by the type graph, T. This is not only more expressive than typed predicates, but can more naturally be apply

existing ontologies, such as KNOWROB [77, 11], RoboEarth [78], AfRob [82] and others [69] when defining world states.

2.1.1.3 The need for transitive (compositional) relationships

Of paramount note is the fact that both representations lack automatic transitive relations which is an implicit relation that could aid in reasoning about indirect object interactions in the scene. Transitive relations enable a planner to reason about indirect relationships between objects in the scene. For example, if object A is on top of object B, and object B is on top of object C, a transitive relation allows the planner to infer that object A is indirectly on top of object C. This can also be thought of as a composition of paths from A to C. Transitive relations in scene graphs would also allow for a more compact and efficient representation of the relationships between objects.

In PDDL, it is possible to incorporate transitive relations using derived predicates in PDDL 2.2 [26], but they cannot be stated universally and they must be defined for each domain. In typed graphs, it is possible to incorporate path equations that establish path equivalences for all composable combination of paths, but this would require inflating edge list to include these additional paths. Both approaches would increase the computational complexity of the search space significantly.

We hypothesize that this structure can help a planner generate more efficient and valid plans that consider such indirect relationships, both taxonomical and associative relations. In this thesis, we propose to demonstrate the value of this property in improving the efficiency and quality of plans in real-world scenarios.

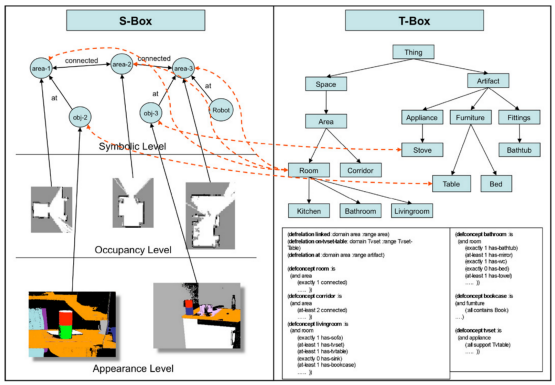
2.1.2 General approaches to integrating semantic knowledge

There has been a concerted effort across robotics to integrate semantic information about the world in planning and control through ontologies and other typing formalisms [56, 58]. Most efforts are directed towards directly translating sensor streams into meaningful predicates for planning [12, 65, 47], establishing default knowledge through axioms and data-driven modeling [78, 79, 57], and converting ontologies to task-relevant predicates [33, 41, 87, 19]. However, there are few examples that directly integrate ontologies to enrich world states for robot task planning based on the widely-accepted formalism, PDDL [36].

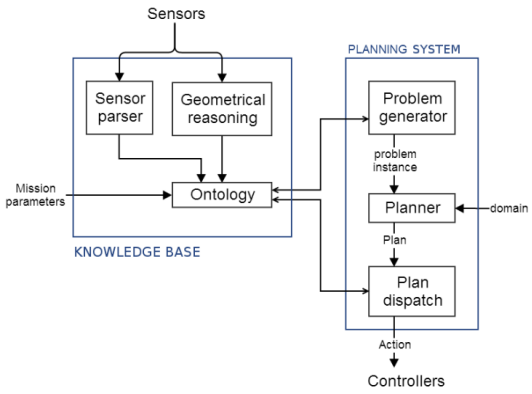
Among the aforementioned, only the work by Galindo et al [33] and Cashmore et al [19] fit this criteria. Cashmore et al [19] designed an framework called ROSPlan that contains two components, a knowledge-base and a planning system. The architecture is illustrated in 2.1(b). The knowledge base includes an ontology subcomponent that is provided by the experts of the domain and can be expressed using the Web Ontology Language (OWL) [10]. Another subcomponent interprets sensor readings from the agent and augments the knowledge base with scene information based on the ontology. To initiate planning, a domain file must be provided by an end user and the initial state is populated based on the knowledge base and according to the specifications of the domain file. Overall, this framework proposes a reasonable high-level architecture, however, the specific details about how information is translated at every interface is not provided, making it difficult to state how this architecture will behave on a range of domains and planning problems.

Galindo et al [33] use a two-part knowledge representation system, which includes (i) spatial information about the robot environment in the form of a scene graph, called S-Box, and (ii) an ontology that describes the hierarchical relationships between concepts, called T-Box. A total function mapping from literal objects in the spatial graph to terminological concepts in an ontology is defined, as shown in Figure 2.1(a). Predicates observed in the S-Box along with predicates observed in the T-Box are translated into PDDL predicates forming the initial state. From here, planning proceeds as normal using an existing typed planner, namely Metric-FF [42]. The benefits of this approach is the ability to make use of the full ontology to add implicit information as predicates to the world state based on the hierarchy supplied by the T-Box. However, after

the information is sent to the planner, the semantic enrichments to the domain via concept axioms are lost. This makes it impossible to ensure that the concepts are preserved throughout the planning process. For example, an action might cause the robot to move a bed from the bedroom to the kitchen, in which case, it is unclear as to whether that should be valid or invalid within the semantic model. The choice is up for debate, however, we believe that the decision of whether th semantic model is violated by an action should be unambigous within the formalism. Our approach aims to preserve relationships declared in the ontological model throughout the planning process.



((a)) Method for relating spatial information, S-Box, and terminological information, T-Box, proposed by Galindo et al [33]



((b)) Architecture for incoporating ontology information into the planner proposed by Cashmore et al [19]

Figure 2.1: Proposed methods for integrating semantic information during task planning

2.2 Integrating scene graphs

The recent invention of scene graphs has led to the development of a few more frameworks that attempt to make efficient use of these descriptive models of the world during task planning. Scene graphs are a specialized version of knowledge graphs that restrict its objects, attributes, and relations to facts obtained through vision-based perception and inference [54, 6, 21]. In scene graphs, ontologies are often used to align scene data to class hierarchies. However, approaches, as in the previous case, are scarce. We have identified a few attempts [2, 64] to make effective use of scene graphs, as formally defined, in task planning.

Agia et al [2] tackles the problem of managing the complexity of scene graphs for task planning. Scene graphs are a useful abstractions for representing an environment perceived by a robot, but are often too complex, with numerous vertices and edges, making them difficult to reason over at scale. To mitigate this, planners can employ procedures that determine which attributes of the scene are most relevant while also preserving the semantics provided by the class hierarchy and object features. An example of a planner designed for this purpose is the SCRUB planner [2]. SCRUB is a planner-agnostic procedure that prunes the state space to include only the relevant facts within a scene graph. It is paired with SEEK [2], a planner-agnostic procedure that scores objects in the scene based on an importance score produced by a graph neural network [72]. All objects that are ancestors to the relevant objects, according to some threshold, are preserved as facts in the state. Both the SCRUB and SEEK procedures dramatically reduce the number of facts needed to characterize the world state. The facts in the world are translated into binary predicates and

passed to a classical planner. This provides a heuristic-based measure for identifying relevant facts which are, like most heuristics, subject to limitations, such as inaccurate approximations. The combinatorial approach we propose uses the existing semantic structure to determine relevance of objects and relationships within the scene.

Miao et al take an alternate approach to using scene graphs to describe world states and action models [64]. In their approach, action operators are specified in terms of an initial state subgraph, a final state subgraph, and an intermediate subgraph. For each object and relation in these subgraphs, the global scene graph is updated by adding objects into the scene graph that are discussed in the action model. An action is applicable if the global scene graph includes objects from the initial subgraph. If objects are referenced in the final state subgraph that are not present in the scene, they are introduced as isolated vertices. These vertices are connected to the global scene graph by consulting an external knowledge base, such as ConceptNet [73], that contains a type hierarchy. Their procedure searches for a matching object type, identifies a parent type that exists in the scene graph, and defines an edge from that object to the existing type. This is a fragile approach to resolving changes in the world state because it relies on an external knowledge base to be correctly and completely instantiated in order for new information to be properly integrated.

Jiao et al [48] presents a method for efficient sequential task planning using a 3D scene graph representation called contact graph+ (cg+). This representation abstracts scene layouts by identifying valid robot-scene interactions. Goal configurations can be naturally specified on contact graphs and produced by a genetic algorithm with a stochastic optimization method. A task plan is then initialized by computing the Graph Editing Distance (GED) between the initial contact graphs and the goal configurations, which generates graph edit operations corresponding to possible robot actions. The strengths of this method include its ability to bridge robot perception and execution by organizing scene entities and predict outcomes of actions using graph edits. They highlighted the fact that robots successfully completed complex sequential object rearrangement tasks that are difficult to specify using conventional planning language like Planning Domain Definition Language (PDDL), demonstrating the high feasibility and potential of robot sequential task planning on contact graph. One limitation of this method is that it not suitable to for non-manipulation tasks. Additionally, the method relies on the availability of accurate 3D scene information prior to constructing the contact graph representation.

Overall, incorporating semantic information into the representation for world states during planning has gained some attention. These methods satisfy the need to support rich and complex representations of the world using ontologies but still suffer from an inability to preserve ontological information within the planning process for a wide range of tasks. In our approach, the ontology is an integral part of the planning formalism, with which comes with specialized tooling for manipulating such data. Additionally, because objects and relations from the ontology are used to identify relevant substructures in the graph, we hypothesize that our approach will be able to scale to large scene graphs without the need for predicting object relevancy.

2.3 Task planning

In this section, we go over the model for planning based on the state transition system. We also discuss some of the common representation languages used in symbolic planning, namely STRIPS and PDDL and their strengths and limitations in expressivity.

2.3.1 State Transition System Model of Planning

The state transition system model of planning is a formal representation of a planning problem that describes the state space of the problem and the possible actions that can be taken to move between states. In this model [37], a *planning problem* can be defined as a tuple $P = \langle S, A, \gamma \rangle$, where:

- The *state space* $S = \{s_0, s_1, s_2, ...\}$ is the set of all possible states. A state $s \in S$ represents a snapshot of the world at a particular point in time. It ideally includes all relevant information about the state of the world, such as the location of objects and their properties.
- The *action space* $A = \{a_0, a_1, a_2, ...\}$ is the set of all possible actions. An action represents a transition from one state to another state.
- The *transition function* $\gamma : A \times S \to S$ is a partial function that, where it is defined, maps an action and a state to the next state.
- An action $a \in A$ is *applicable* at state $s \in S$ if $\gamma(a, s)$ is defined. A *plan*, $\pi = \langle a_1, a_2, \ldots, a_n \rangle, a_i \in A$, is any sequence of actions. It is a *solution* to the planning problem if it transitions from the initial state s_0 to the goal state s_g, i.e., $\gamma(a_i, s_{i-1}) = s_i$ for $i = 1, \ldots, n$ and $s_n = s_g$.

The state transition system model of planning is often used in automated planning systems, which use search algorithms to explore the state space and find a sequence of actions that achieves the goal. The search algorithms typically use heuristics to guide the search and improve its efficiency. The state transition system model of planning provides a structured and formal way to represent planning problems and reason about the possible sequences of actions that can be taken to achieve a goal. The manner in which states, actions, and transition functions are represented both classically and categorically is a focal comparison in this thesis.

2.3.2 STRIPS (Stanford Research Institute Problem Solver)

STRIPS (Stanford Research Institute Problem Solver) is a planning formalism that was developed in the late 1960s and early 1970s by Richard Fikes and Nils Nilsson [28] at the Stanford Research Institute. STRIPS is an early example of an automated planning system and is often used as the basis for other more complex planning systems. The STRIPS planning formalism adheres to the state transition system model and represents the world as a set of states. The planning problem involves finding a sequence of actions that transforms an initial state into a goal state.

In STRIPS, the world is represented as a set of propositions or predicates. Actions are defined with preconditions, which must be satisfied for the action to be executed, and effects, which describe the changes that occur in the world when the action is executed. The effects of an action are represented as two sets: the add list and the delete list. The add list contains the predicates that become true when the action is executed, while the delete list contains the predicates that become false. The planning process in STRIPS consists of searching through the space of possible action sequences to find one that starts from the initial state and reaches a state that satisfies the goal conditions. This is typically accomplished using search algorithms, such as depth-first search or best-first search, combined with heuristics to guide the search process.

More formally, STRIPS can be represented as a state-transition system, as discussed in Section 2.3.1. A STRIPS planning problem can be formulated using the following tuple [28]: STRIPS $= (P, A, I, G)$

- P: A finite set of predicates that describe properties of the world state. The world state is represented as a set of ground literals, which are instantiations of these predicates.

- A: A finite set of actions, where each action a is represented as a tuple (pre(a), add(a), del(a)):
  - pre(a): Precondition list, a set of ground literals that must be true in the current state for the action to be applicable.
  - add(a): Add list, a set of ground literals that become true after the action is executed.
  - del(a): Delete list, a set of ground literals that become false after the action is executed.
- I: The initial state, which is a set of ground literals representing the properties that hold true at the beginning of the planning process.
- G: The goal, which is a set of ground literals representing the properties that must hold true in the final state after the plan is executed.

A solution to a STRIPS planning problem is a sequence of actions that, when executed in the initial state I, leads to a state where all the literals in the goal G hold true. The planning process involves finding such a sequence of actions that satisfy the goal while respecting the preconditions and effects of the actions. STRIPS planning formalism laid the foundation for many subsequent planning representations, such as ADL (Action Description Language) and PDDL (Planning Domain Definition Language), which extended and improved upon the original STRIPS concepts.

2.3.3 PDDL (Planning Domain Definition Language)

PDDL (Planning Domain Definition Language) is a language that provides an extension to STRIPS that supports a richer set of features, such as typed variables, temporal planning, and numerical fluents [36]. A PDDL problem specifies actions, objects, and goals, as well as the initial state of the problem. Currently, PDDL is the most widely used specification language for planning problems and domains. While PDDL seeks to represent the planning problem as a state-transition system, a formal semantics that pairs with its syntax: there is no precise mathematical definition of its meaning [30]. In any case, there does exist a formal definition of the PDDL syntax. The PDDL language describes planning problems in a domain-independent way, separating the domain description from the problem description [37].

A PDDL domain description, based on PDDL1.2 [51],consists of:

- A finite set of types, which define the categories of objects in the domain.
- A finite set of predicates, which describe properties of objects and their relationships.
- A finite set of actions, where each action is specified with parameters, preconditions, and effects.

A PDDL problem description consists of:

- A finite set of objects, which belong to the types defined in the domain description.
- An initial state, which is a set of ground literals (instantiations of predicates) describing the properties of the objects that hold true at the beginning of the planning process.
- A goal, which is a set of ground literals representing the properties that must hold true in the final state after the plan is executed.

An example domain and problem file for a domain designed for Blocksworld can be found in Appendix A.

PDDL also includes numerous language extensions that add useful features to PDDL including preferences [35], temporal constraints [66], and support for negative literals [30]. While these extensions make PDDL a more powerful and flexible language for specifying planning problems and domains, they also introduce additional complexity and possibilities for inconsistency when used with planning algorithms [45] as its not clear how these extensions universally impact the search space. Therefore, developing a language with clear and consistent formal semantics in planning is crucial for improving the performance of planners. In our approach, we aim to provide formal semantics for transitive closure and typing within the planning process to improve the overall plan quality.

2.3.4 Assessing Plan Quality

Plan quality as an evaluation metric can be broken down into a number of features about the plan and the planning process. Some common criteria used to assess plan quality in AI planning include [45]:

- *Plan length*: This refers to number of sequential steps in a plan that achieve a goal.
- *Efficiency*: This refers to how quickly the plan can be executed and how well it makes use of available resources.
- *Success Rate*: This refers to whether the plan is able to achieve the desired goal in a variety of possible scenarios.
- *Stability*: This refers to how robust plans are in light of syntactic or semantic changes in the environment [29].
- *Flexibility*: This refers to how easily the plan can be adapted to changing circumstances or new information.

These criteria can be used individually or in combination to assess the quality of a plan generated by a planning system. All of the above criteria would be useful for assessing plan quality in our approach, however, we will omit measuring *flexibility* of plans generated as this would require substantial effort to create an effective adaptation algorithm within the scope of this project. This is left for future work.

2.3.4.1 Experiment set-up

Howe and Dahlman [45] summarize the canonical experiment set-up for benchmarking planning algorithms as shown in Figure 2.2. This will be more or less adhered to in the evaluation of our approach. The details are discussed in Chapter 6.

2.4 Category Theory

In this section, we will motivate the cause for using category theory, provide definitions that serve as the necessary foundations to category theory, summarize the related work.

Eilenberg and MacLane [27] introduced the concepts of category theory in the mid-1940s during their study of algebraic topology as a way to transfer theorems between algebra and topology. In doing so, they provided a mathematical language that lifts many mathematical and non-mathematical concepts to the notion of maps between entities and compositions of those maps. It provides a mathematical framework

1. Select and/or construct a subset of planner domains
2. Construct problem set by:
   - running large set of benchmark problems
   - selecting problems with desirable features
   - varying some facet of the problem to increase difficulty (e.g., number of blocks)
3. Select other planners that are:
   - representative of the state of the art on the problems OR
   - similar to or distinct from the new planner, depending on the point of the comparison or advance of the new planner OR
   - available and able to parse the problems
4. Run all problems on all planners using default parameters and setting an upper limit on time allowed
5. Record which problems were solved, how many plan steps/actions were in the solution and how much CPU time was required to either solve the problem, fail or time out

Figure 2.2: A canonical experiment setup for evaluating planners described by Howe and Dahlman [45]

that aids in understanding the structure of, and relationships between, different mathematical objects and places a particular emphasis on preserving composition in higher-order mappings. There has also been considerable effort in recent years to employ such structures in defining a generic framework for consist and formal manipulation of ontology-backed databases [18, 68, 75, 67, 74].

It has also found important applications in a variety of fields, including systems engineering and design [16, 17, 20], model-driven engineering for software system design [53, 24], and physics [8, 1]. Category theory has been applied to engineering and scientific problems in various domains such as database migration [76, 74]; heterogeneous information and sensor fusion [70, 49, 52]; scientific knowledge representation [40]; and automated assembly planning [15]; however, there are few compelling examples in the field of artificial intelligence. The few examples in robotics and planning [3, 4, 63] illustrate some promising use cases, but none have been tested on a wide range of problems at meaningful scales.

In this thesis, we seek to extend the application of category theory to AI task planning to address the problem of ensuring high quality plans over complex domains.

In Chapter 4, we claim that this can be done by viewing the symbolic planning problem as a series of structure-preserving graph rewrites over typed graphs.

2.4.1 Core concepts of category theory

Prior to discussing the formal definitions, we would like to provide an introductory exposition to category and its core concepts. When approaching a new mathematical language, it is helpful to remember that such structures are languages that supply the types and grammar that can be used to discuss a given concept. For example, graphs possess only two types: (i) nodes and (ii) edges. They also have only one grammar rule, that being: an edge can only exist between at most two nodes. Much can be said using these types and grammar rules, but also that is much that needs to be translated to other structures to model more complex concepts.

Category theory defines the following types: (i) *objects* and (ii) *morphisms*. At a glance, the cardinality of types in graphs and categories seem to imply that they have equal expressive power. However, categories

are equipped with a laundry list of grammar rules that allow for a richer set of constructions. This list provides a foray into the foundational grammar rules.

- *Morphisms* are directed relations, like arrows, that exist between at most two objects, a source and a target.
- *Morphisms* can be *composed* if the target of the precomposed arrow is equal to the source of the postcomposed arrow.
- This operation is associative.
- There exists a special arrow called the *identity arrow* for every object that sends the object to itself
- *Categories* consist of a collection of objects and morphisms.

This provides the basic rules for a category. The most prominent feature of this language is the notion of composition of morphisms. Because of the simplicity and generality of these definitions, this gives freedom to embed more complex structures within each concept while still maintaining closure within the categorical language. For example, it is possible to consider a category on its own as an object and a morphism between categories as objects as a *functor*. A functor is defined as a map between objects and morphisms of categories such that the composition of morphisms is preserved on both sides of the map. It is also possible to describe further restrictions on the definitions of objects and morphisms, such as a morphism between categories is a functor that only maps objects to objects and ignores arrows, namely a *bijection-on-objects functor*. Another example might be that we want to restrict objects in a category to only consist of sets and arrows to be functions– which forms the *Category of Sets*.

To summarize and expound, category theory revolves around a few fundamental concepts that serve as the building blocks for its applications in computer science [55]:

- Categories: A category consists of objects and morphisms (arrows) between those objects, satisfying specific properties. Objects can represent different types of entities, while morphisms represent the relationships or transformations between objects.
- Functors: A functor maps a category to another category, preserving the structure of the original category. Functors translate objects and morphisms between categories while maintaining their relationships. They can also be viewed as a way to construct categories from other ones.
- Natural Transformations: A natural transformation is a higher-order structure that relates two functors. It provides a way to compare and transform functors while preserving their underlying structure.
- Limits and Colimits: Limits and colimits are universal properties in category theory that capture the essence of various types of aggregation and decomposition of objects and morphisms.

2.4.2 Categories, functors, natural transformations

A more formal definition of a category can be seen in Definition 2.4.1.

Definition 2.4.1 (Category)**.** A *category* C consists of a collection of *objects*, denoted $\mathrm{Ob}(\mathsf{C})$; for every pair of objects $x, y \in \mathrm{Ob}(\mathsf{C})$, a collection $\mathrm{Hom}_{\mathsf{C}}(x, y)$ of *morphisms from x to y*, whose elements $f \in \mathrm{Hom}_{\mathsf{C}}(x, y)$ are denoted $f : x \to y$; a *composition* operation, defining for each pair of morphisms $f : x \to y$ and $g : y \to z$, a *composite* morphism $g \circ f : x \to z$; for every object x, an *identity* morphism $1_x : x \to x$; satisfying the

associativity law $h \circ (g \circ f) = (h \circ g) \circ f$ and *unitality* laws $f \circ 1_x = f$ and $1_y \circ f = f$ whenever these equations make sense.

Categories are themselves the objects of a category, whose morphisms are called *functors*. A functor consists of compatible maps between the objects and between the morphisms that preserve composition and identities.

Definition 2.4.2 (Functors)**.** A *functor* F from a category C to another category D, denoted $F : \mathsf{C} \to \mathsf{D}$, consists of a map between objects $F : \mathrm{Ob}(\mathsf{C}) \to \mathrm{Ob}(\mathsf{D})$, and for every pair of objects $x, y \in \mathsf{C}$, a map between hom-sets $F : \mathrm{Hom}_{\mathsf{C}}(x, y) \to \mathrm{Hom}_{\mathsf{D}}(F(x), F(y))$, such that the following equations hold:

- $F(g \circ f) = F(g) \circ F(f)$ for every $x \xrightarrow{f} y \xrightarrow{g} z$ in C;
- $F(1_x) = 1_{F(x)}$ for every $x \in \mathsf{C}$.

There can also exist morphisms between functors, called natural transformations.

Definition 2.4.3 (Natural Transformation)**.** Let C and D be categories, and let F and G be functors from C to D. A *natural transformation* η from F to G is a family of morphisms in D, indexed by the objects of C, such that for each object X in C, there is a morphism $\eta_X : F(X) \to G(X)$ in D, satisfying the following naturality condition: for every morphism $f : X \to Y$ in C, the following diagram commutes in D:

$$\begin{array}{ccc} F(X) & \xrightarrow{\eta_X} & G(X) \\ \downarrow F(f) & & \downarrow G(f) \\ F(Y) & \xrightarrow{\eta_Y} & G(Y) \end{array}$$

2.4.2.1 Some notable functors and natural transformations

Of the many constructions that can be performed using categories and functors, we mention just a few that are used in the thesis.

Definition 2.4.4 (Grothendieck Construction, Category of Elements)**.** Let $F : \mathsf{C} \to \mathsf{Set}$ be a functor from a small category C to the category of sets and functions. The *category of elements* of F, denoted $\int F$, is the category whose

- objects are pairs (c, x), where $c \in \mathrm{Ob}(\mathsf{C})$ and $x \in F(c)$;
- morphisms from (c, x) to (d, y) are morphisms $f : c \to d$ in C such that $F(f)(x) = y$.

Composition and identities in the category of elements are inherited from those in C [55].

Lemma 2.4.1 (Yoneda Embedding)**.** The *Yoneda embedding* is the full and faithful functor $\mathsf{C}^{\mathrm{op}} \to [\mathsf{C}, \mathsf{Set}]$ that sends:

- (objects) $c \mapsto H^c$, where $H^c(d) := \mathrm{Hom}_{\mathsf{C}}(c, d)$ is the representable functor;
- (morphisms) $(c \xrightarrow{f} d) \mapsto (H^d \xrightarrow{H^f} H^c)$, where H^f is the natural transformation between representable functors given by precomposition with f.

Definition 2.4.5 (Diagram, D)**.** Let C be a category and I be an index category. A *diagram* D of shape I in the category C is a functor $D : \mathsf{I} \to \mathsf{C}$. In other words, a diagram assigns an object in C to each object in I and a morphism in C to each morphism in I, such that the following conditions hold:

1. For every object i in I, there is an object $D(i)$ in C.
2. For every morphism $f : i \to j$ in I, there is a morphism $D(f) : D(i) \to D(j)$ in C.
3. For every pair of composable morphisms $f : i \to j$ and $g : j \to k$ in I, $D(g \circ f) = D(g) \circ D(f)$.
4. For every object i in I, $D(\mathrm{id}_i) = \mathrm{id}_{D(i)}$.

A diagram in category theory can be thought of as a functor that assigns an index category, I, to a substructure of another category, C.

Span A *span* is a diagram that will appear several times throughout this thesis. It it a diagram with an index category containing three objects and two morphisms arranged like

$$X \xleftarrow{f} Z \xrightarrow{g} Y$$

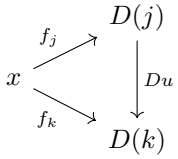
f is called the *left leg* of the span and g is called the *right leg* of the span. Z is called the *apex* of the span.

2.4.3 Universal properties

Objects in categories can satisfy universal properties such as having limits or colimits [55]. Limits and colimits are taken over diagrams in a category. Let J be a small category. A *diagram of shape* J in a category C is a functor $D : \mathsf{J} \to \mathsf{C}$.

Definition 2.4.6 (Limit)**.** Let $D : \mathsf{J} \to \mathsf{C}$ be a diagram.

- A *cone* over D is an object $x \in \mathsf{C}$ and a family of arrows in C, $(x \xrightarrow{f_j} D(j))_{j \in \mathsf{J}}$, such that the triangle



  commutes for every morphism $u : j \to k$ in J.

- A *limit* of D is a cone $(L \xrightarrow{\pi_j} D(j))_{j \in \mathsf{J}}$ over D with the property that for any cone over D as above, there exists a unique map $f : x \to L$ such that $\pi_j \circ f = f_j$ for all $j \in \mathsf{J}$.

Colimits are defined dually to limits.

Definition 2.4.7 (Colimit)**.** Let $D : \mathsf{J} \to \mathsf{C}$ be a diagram.

- A *cocone* under D is an object $x \in \mathsf{C}$ and a family of arrows in C, $(D(j) \xrightarrow{f_j} x)_{j \in \mathsf{J}}$ such that the triangle

$$\begin{array}{ccc} D(j) & & \\ \downarrow{\scriptstyle Du} & \searrow^{f_j} & \\ D(k) & \nearrow_{f_k} & x \end{array}$$

  commutes for every morphism $u : j \to k$ in J.

- A *colimit* of D is a cocone $(D(j) \xrightarrow{\iota_j} C)_{j \in \mathsf{J}}$ under D with the property that for any cocone under D as above, there exists a unique map $f : C \to x$ such that $f \circ \iota_j = f_j$ for all $j \in \mathsf{J}$.

Colimits can be obtained over a variety of diagrams. Of relevant note to this thesis, is the colimit over a span, defined earlier. A colimit of a span is called a pushout and it forms a pushout square.

Definition 2.4.8 (Pushout)**.** Let A be a category, and take objects and maps $\begin{array}{ccc} Z & \xrightarrow{s} & Y \\ \downarrow{\scriptstyle t} & & \\ X & & \end{array}$ in A. A *pushout* of this diagram is an object $P \in \mathsf{A}$ with maps $p_1 : X \to P$ and $p_2 : Y \to P$ such that $\begin{array}{ccc} Z & \xrightarrow{s} & Y \\ \downarrow{\scriptstyle t} & & \downarrow{\scriptstyle p_1} \\ X & \xrightarrow{p_2} & P \end{array}$ commutes.

Pushout in Set For the category of Set, a pushout can be thought of as a disjoint union joined along the set in the apex of the span.

2.4.4 Existing applications of category theory in computer science

In addition to the aforementioned list, we wish to highlight applications of category theory to the field of computer science, including: programming languages, type systems, domain-specific languages, and formal verification. In the case of programming languages, category theory has influenced the design of functional programming languages, such as Haskell, by providing concepts like monads, functors, and arrows [81]. These abstractions enable powerful and expressive ways of structuring programs and reasoning about their behavior. The field of homotopy type theory has also benefited from category theory concepts by using it to develop more expressive and flexible representations of data structures and algorithms [60, 7].

The most relevant applications of category theory to this work are those related to data modeling, knowledge representation, and graph transformation systems. Category theory has been used to model and reason about complex data structures and knowledge bases, providing a means for expressing and manipulating structured information. This is most notably found in Spivak [76, 74] and Pattersons [67] exploration of ontology-logs, or o-logs, and functorial data migration. In these works, category theory has been used to model and reason about ontologies, providing a mathematical framework for expressing and manipulating structured information. It has also been shown to be interoperable with resource description format (RDF), a semantic web standard data format [74]. In the context of databases, category theory has been utilized for schema mapping and integration. Functors can be used to represent the transformation of

data between different schemas, while natural transformations can express the relationships between different schema mappings. This enables a systematic approach to integrating and transforming data across diverse databases.

2.4.4.1 Functorial data migration

Functorial data migration is an approach rooted in category theory that offers a structured and rigorous framework for transforming and migrating data between different schemas introduced by Spivak in 2012 [74]. Here Spivak presents a framework for representing database schemas and instances using category theory. In this framework, a database schema is represented as a small category, where the objects of the category represent the tables in the schema and the morphisms represent the relationships between the tables. An instance of the schema is then represented as a set-valued functor on the category, where the functor assigns to each object (table) a set of rows and to each morphism a function that represents the relationship between the rows of the corresponding tables.

The paper shows that given a morphism between two schemas (represented as categories), there are three "data migration functors" that can be used to translate instances from one schema to another in a canonical way. These functors parameterize projections, unions, and joins over all tables simultaneously and can be used in place of conjunctive and disjunctive queries.

Overall, this framework provides a rigorous and mathematically well-founded approach to representing and manipulating database schemas and instances using category theory. This framework can be adapted to our problem by treating the desired ontology as schema category and the instances of classes in the ontology, obtained through perceptual means, as the sets. More details about this implementation can be found in Chapter 3.

2.4.4.2 Graph rewriting systems

Category theory has contributed to the development of graph transformation and rewriting systems, offering a general framework for expressing and analyzing complex graph-based algorithms and data structures [18]. In particular, category theory has been used to formalize and analyze graph transformation systems, where graphs are used to represent structures and transformations between graphs represent computational steps. The Double-Pushout (DPO) approach is a widely-used category-theoretic formalism for graph transformation, based on the concepts of pushouts and pullbacks, which allows for a precise and compositional description of graph rewriting rules. In this context, category theory provides a way to model and reason about the structure and behavior of term rewriting systems, offering insights into their properties and correctness. More details about this is applied to planning can be found in Chapter 4.

In summary, category theory provides a versatile and powerful framework for understanding and modeling abstract structures and relationships in computer science. Its applications span a wide range of areas, including programming languages, systems engineering, physics, and scientific modeling, enabling the development of more expressive, flexible, and rigorous software systems and tools. However, even more relevantly, category theory has been applied to various aspects of data modeling and knowledge representation, such as ontologies, database schema mapping, and knowledge representation formalisms. By leveraging the expressive power and compositional nature of category theory, these applications benefit from a rigorous and flexible framework for reasoning about complex relationships, structures, and algorithms. Throughout this thesis, we will draw heavily on these branches of application.

Chapter 3

Preliminary Result I - C-set for representing complex world states

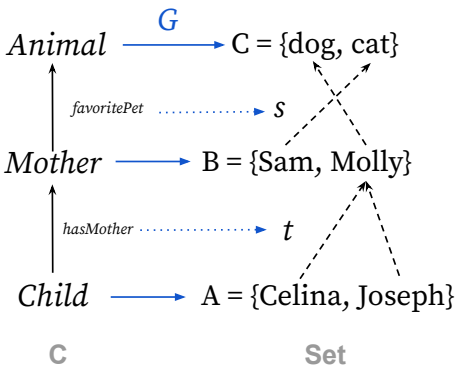
In this chapter, we will cover preliminary work that demonstrates how C-sets from category theory can be used to represent typed graphs. This is a useful formalism for encoding complex world states containing many objects and relations.

3.1 Our approach: Using C-sets

To retain the benefits of typed graphs while incorporating transitive relations, otherwise known as compositional relations, we propose the use of the C-set formalism introduced by Spivak in 2012 [74]. This is possible because all typed graphs can be expressed as C-sets according to [18]. The key feature of this formalism is its ability capture the relationship between logical syntax (theories) and semantics (models) while preserving compositional structure using functors. This paradigm is known as *functorial semantics* [61]. This differs from the standard formalism of first order logic because it gives the syntax of the language, which we can consider the ontology, as an algebraic object independent of its semantics. An illustration of a C-set can be found in Figure 3.1.

This representation provides a denotational semantics for typed graphs using the category C-Set. Let C denote a small category, called a *schema*. A C-*set*, also known as a *copresheaf on* C, is a functor from C to the category Set. The schema is a category whose objects we interpret as types and whose morphisms describe "is-a" and other functional relationships between types. The category Set is the category of sets and functions that is providing the instance data, or semantics, about the world state. Thus, a C-set is a functor that sends types to sets and type relationships to functions. On this interpretation, C-sets are a simple but useful model of relational databases [75]. It is important to note that the sets are related by functions and not relations. This adds additional structure to the nature of the relationships that exist between instances using C-sets. This restriction is necessary to ensure that compositional, or transitive, relationships are well-defined.

The category of elements (see Definition 2.4.4) of a C-set X, denoted $\int X$, packages the data of X into a category resembling a knowledge graph. Specifically, a morphism in the category of elements can be interpreted as a Resource Description Framework (RDF) triple [67], which is a common text-based serialization format for knowledge and scene graphs. A C-set is depicted in this style in Figure 3.1. A formal definition of a C-set is provided in Definition 3.1.1.




```
rdf(Sam::Mother, s::favoritePet, cat::Animal)
rdf(Molly::Mother, s::favoritePet, dog::Animal)
rdf(Celina::Child, t::hasMother, Molly::Mother)
rdf(Joseph::Child, t::hasMother, Molly::Mother)
rdf(Celina::Child, t;s::hasMother;favoritePet, dog::Animal)
rdf(Joseph::Child, t;s::hasMother;favoritePet, dog::Animal)
```

Figure 3.1: An example C-set, G, that stores data about children and their mother's and their mother's favorite pet. The category of elements contains triples analogous to RDF triples.

Definition 3.1.1 (Category of C-sets, C-Set)**.** For a given schema C, the *category of* C*-sets* is the functor category C-Set := $\mathsf{Set}^{\mathsf{C}}$, whose objects are functors from C to Set and whose morphisms are natural transformations between those.

The category of C-sets is a topos, an especially well-behaved kind of category in which, for example, all limits and colimits exist. This will be a useful fact when describing how we transform world states using actions.

3.1.1 Morphisms between C-sets

For rewriting systems, typically morphisms by C-sets are supplied. These morphisms contain considerable amount of data, given that they must match components of functors to one another such that they preserve composition. In practice, this would not be a useful procedure to incorporate into task planning or in any case. Therefore, a generic morphism matching algorithm was introduced by Brown et al [18] which can be thought of as a type of structural pattern matching.

The problem of finding a morphism, $m : X \to Y$, between two C-sets, X and Y, can be translated into a typed constraint satisfaction problem (CSP). This is done by taking the elements of the two C-sets and using them as the variables and domain of the CSP. Recall that in a constraint satisfaction problem (CSP), variables are the elements of the problem that can take on different values. The domain of a variable is the set of all possible values that the variable can take on. The goal of a CSP is to assign a value to each variable such that all constraints are satisfied [71]. The types in the CSP are given by the objects of the indexing category C. For each morphism, $f : c \to c'$, in C and each element, given by $X(c)$, a constraint is introduced.

The solutions to this CSP are exactly the C-set homomorphisms between the two C-sets which are natural transformations [18].

This means that the algorithm only considers assignments that satisfy the typing relations. The typed CSP search space grows by $\mathcal{O}(n^k)$ where n is the size of the target (Y) and k is the size of the source (X) [18]. For reference, a generic graph homomorphism matching problem is NP-complete, making this a more tractable algorithmic problem to solve than general graph homomorphism matching.

3.1.2 Benefits and limitations of using C-sets

C-sets offer several benefits for planning. By leveraging C-sets, we can preserve composition, which allows for effective modeling of intricate relationships. Additionally, C-sets enable the expression of negative applicability conditions, which are crucial in representing constraints and restrictions that impact the dynamics of the world. This will be discussed in Section 4.1.3. They also support quantitative attributes and their evaluation, such as addition, subtraction, multiplication, and other arithmetic operations. Lastly, the functional requirement for relation instances is helpful in guarding against contradictory predicates such objects being at two places at once which may be possible to express in STRIPS as `on(b1 - Block, table - Table)` and `on(b1 - Block, floor - Surface)`.

At the moment, using C-sets to model world states comes with some limitations. One potential challenge is the need to consider automatic transitive closure when designing the ontology, while beneficial, can be a non-intuitive process. Additionally, the connection between C-sets and motion planning has not been explored, which is vital for certain applications like robotics or autonomous vehicles. C-sets also do not support ordered comparisons, such as less than or greater than relations, limiting their expressiveness when comparing attributes or evaluating specific conditions. Lastly, this formalism currently only support binary relations between objects; whereas, in formal ontologies, it is possible to express n-ary relations.

In summary, while C-sets offer numerous benefits for modeling world states, it is important to be aware of their limitations and consider alternative methods or augmentations if necessary, depending on the particular application. An implementation of C-sets can be found in the Julia package Catlab.jl. An example of a world state defined using C-sets can be found in Appendix B.1.2.1 and B.1.2.2.

Chapter 4

Preliminary Result II - Planning using C-set and DPO rewriting

In this chapter, we discuss the preliminary work done to describe a planning language based C-sets and double-pushout (DPO) rewriting. Aside from being expressive enough to execute planning over typed graphs, it also provides the additional benefit of preserving compositional relationships found in the schema category, or ontology. This chapter presents an example of how an action in the proposed representation differs from actions in the STRIPS-PDDL representation.

In particular, we propose a change to scene graph planning architectures that instead lifts components of the framework to the shared language we will discuss in this chapter. Figure 4.1 illustrates the shift from a multi-language architecture for planning over scene graphs to a single language architecture. To understand the limitations of existing work in this space, we refer the reader to Chapter 2 for a review. Our proposed architecture eliminates the need for translating into an alternate planning representation and instead consider plans as a sequence of rewrite rules applied to the scene graph.

4.1 Our approach: Using C-sets and DPO rewriting

We propose using C-sets and a procedure for adapting world states called double-pushout (DPO) rewriting in a way that aligns with the state-based transition system planning model described in Section 2.3.1.

4.1.1 Action as Spans in C-Set

Action rules are specified as spans in C-Set. Action rules are made up of components similar to those of action schemas in classical representation. Specifically, actions rules are spans $(I \hookleftarrow K \rightarrow O)$ in C-Set that consists of the *precondition*, I, on the left-hand side, the *effects*, O, on the right-hand side, and the *glue*, K, in the middle, which gives the data that remains unchanged between the input and the output. During implementation, we choose to present spans of C-sets as colimits of representable functors. This conversion is possible because, for every C-set, F, in the action rule, a natural transformation exists from F to a representable functor on C, as per the Yoneda embedding (see Lemma 2.4.1). Covariant representable functors, as defined in Definition 4.1.1, map objects, $A \in \mathsf{C}$ to the set of morphisms that have A as its source object. This conversion also provides us with the the added benefit of matching the implicit substructure when an object, such as A, is explicitly identified.

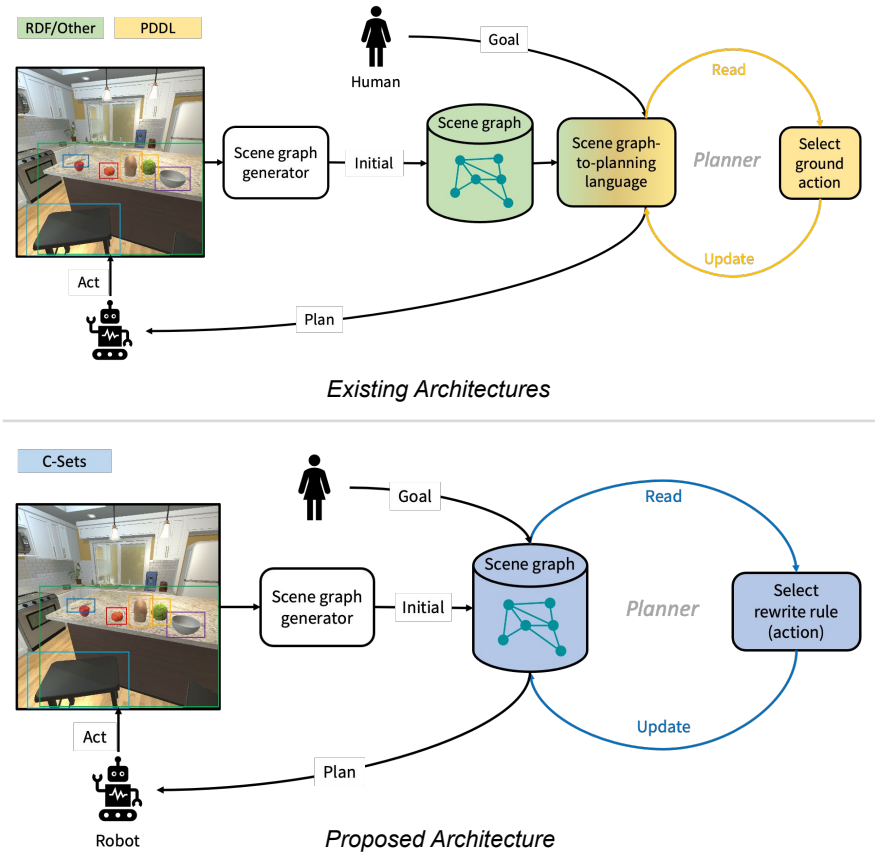


Figure 4.1: A comparison of scene graph architectures and our proposed architecture. The existing architectures summarized here are [64, 33, 2, 19]

Definition 4.1.1 (Representable Functors)**.** A (covariant)[1] representable functor, $H^A : \mathsf{C} \to \mathsf{Set}$, is functor that sends: (objects) $x \in \mathsf{C}, x \mapsto \mathsf{C}(A, x)$ (morphisms) $x \xrightarrow{f} y \mapsto \mathsf{C}(A, x) \xrightarrow{H^A(f)} \mathsf{C}(A, y)$, where $\mathsf{C}(A, x)$ is the set of all morphisms from A and x in the category C. $H^A(f)$ postcomposes f with morphisms in $\mathsf{C}(A, x)$. The functor H^A can also be written as $[\mathsf{C}, \mathsf{Set}]$.

The categorical rule specification differs from the classical one in that it takes a declarative approach by not articulating what atoms should be added and removed from the state, but rather discussing what should be in the state and resolving conflicts using the double-pushout (DPO) rewriting procedure which is discussed in Section 4.1.3.

[1] A representable functor can defined from a covariant ($H^A : \mathsf{C} \to \mathsf{Set}$) or a contravariant ($H_A : \mathsf{C}^{op} \to \mathsf{Set}$) view [61].

4.1.2 Applicability using Monomorphisms

Recall that in classical planning, a precondition is satisfied by a world state when its propositions are a subset of the world state or if there exists a logical entailment between the precondition and the world state. In the category-theoretic context, these notions are generalized via monomorphisms. Monomorphisms generalize the concept of an injective function to arbitrary categories. In Set, monomorphisms are precisely injective functions. In C-Set, monomorphisms are natural transformations such that every component is an injective function, e.g., a monomorphism between graphs is a graph homomorphism such that the vertex and edge maps are both injective. The monic condition (see Definition 4.1.2) applied to morphisms is relevant for applicability because it checks that two entities in the precondition cannot be mapped to the same entity in the world state.

Definition 4.1.2 (Monomorphism)**.** A morphism $f : x \to y$ in a category C is a *monomorphism*, or *monic* for short, if for any other object z and any pair of morphisms $g_1, g_2 : z \to x$, we have $g_1 = g_2$ whenever $f \circ g_1 = f \circ g_2$.

$$z \underset{g_2}{\overset{g_1}{\rightrightarrows}} x \xrightarrow{f} y$$

In other words, a morphism f is monic if whenever two morphisms have the same post-composite with f, then they must be equal.

As mentioned earlier, additional objects and morphisms may be identified beyond what is explicitly expressed in the rule because it is implemented using representable functor. The expanded structure serve as additional constraints, therefore, limiting the number of applicable rules for a given world state.

4.1.3 Transition using Double-Pushout (DPO) Rewriting

The novel insight we provide is that action rules are exactly double-pushout (DPO) rewriting rules. Double-pushout (DPO) rewriting is a type of graph rewriting that is particularly well-suited for algebraic approaches to graph transformation. In fact, DPO rewriting generalizes directly from graph rewriting to C-set rewriting [18]. The DPO method, described below, is used to compute, for a given rule, all possible matches of world with the preconditions and determine which matches are compatible with the effects. The result is a set of transformation steps that can be applied to the target graph.

$$\begin{array}{ccccc} I & \xhookleftarrow{l} & K & \xrightarrow{r} & O \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle f} & & \downarrow \\ X & \xleftarrow[g]{} & Z & \longrightarrow & Y \end{array}$$

DPO rewriting relies on the fundamental concept of a pushout. A pushout is a colimit (See Appendix 2.4.7) of a diagram having the shape of a span ($\bullet \leftarrow \bullet \rightarrow \bullet$). Given a span $R \leftarrow Q \rightarrow S$, a pushout produces an object that resembles the union of R and S joined along Q, $(R \cup S)/Q$. A pushout in C-Set is computed by taking the disjoint union of the sets being pushed out, adding the relations between sets based on C, and quotienting by Q. Pseudocode for the DPO rewriting procedure is given in Algorithm 1. The first step is to find a monomorphism, m, that matches I in X, as described in the previous subsection. The pushout complement, f, is computed provided the morphisms l and m. A pushout complement is a map that manages the deletion of entities that form the complement K/I. Because i is a monomorphism and we assume the identification and dangling conditions [18] are met, the pushout complement exists and is unique up to isomorphism. Having constructed the three sides of the square, l, m, f, the pushout square can be

Algorithm 1 Double-Pushout (DPO) Rewriting

Require: (action rule) $I \hookleftarrow K \rightarrow O \in$ C-Set
Require: (world) $X \in$ C-Set
$m =$ `FindHomomorphism`(I,X)
$f =$ `ComputePushoutComplement`$(l,\ m)$
$g =$ `CompletePushout`$(l,\ f,\ m)$
$Y =$ `ComputePushout`$(f,\ r)$

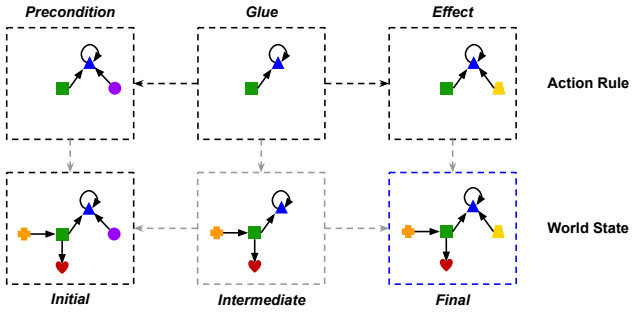


Figure 4.2: An illustration of how DPO rewriting is executed on a C-set where C defines the shapes and the arrows between them. Each shape in the figure is assigned to a color to help with readability.

Representation	Categorical	Classical
State, S	Object in category C-Set	Conjunction of propositions
Action, A	Spans in C-Set containing: Preconditions, Glue, Effects	Action model containing: Parameters, Preconditions, Effects
Applicability	Monomorphisms in C-Set	Subset inclusion
Transition, γ	DPO rewriting	Set-based addition and subtraction

Table 4.1: A summary of the differences between the classical representation and the categorical representation aligned to the state transition system model for planning

completed by a unique map g. Then, to compute the new world state, the right pushout square is computed provided f and r.

The complexity of the `FindHomomorphism()` subroutine is $\mathcal{O}(n^k)$ where k is the size of I and n is the size of the relevant substructure in X which is dictated by the objects in I [18] as discussed in Section 3.1.1. The complexity of the remaining subroutines is the same as that of computing pushouts in Set which is $\mathcal{O}(p)$ where p is the sum of the sizes of the sets involved in the span.

As an example, Figure 4.2 shows how an action rule changes the initial state to the final state. In this figure, the rule states that the circle in the initial state should be replaced by a trapezoid and the square and triangle should persist. The initial state, bottom-left, satisfies the precondition because it contains a matching pattern involving the square, triangle, and circle. This means that a monomorphism can be

identified from the precondition to the initial state. The intermediate state is the result of identifying a pattern that, when joined with the precondition along the glue, produces the initial state. This removes the circle from the initial state. The final state is then constructed by taking the pushout for the span (Intermediate ← Glue → Effect). This produces a pattern where a trapezoid is added in place of the original circle. The example shown in Figure 4.2 demonstrates that rules can involve both the creation and destruction of entities in the world if the precondition contains an entity and the effect does not contain that entity. This allows for a non-monotonicity in updates of the world state.

Handling negative conditions The categorical representation is capable of handling constraints in the form of negative preconditions and effects using DPO rewriting with negative application conditions (NACs) [39]. NACs are a way of restricting the application of an action rule by specifying conditions that must not be present in the world state before or after the rule is applied. In other words, a negative application condition specifies a set of patterns that must not match any part of the world state before or after the rule is applied. The use of NACs in action rules allows for more precise and flexible specification of handling negative preconditions and effects.

4.1.4 An example: moving bread slices

In this section, we discuss the differences between the classical, STRIPS-based, and categorical representations. The comparison is summarized in Table 4.1. We also walk through an example, shown in Figure 4.3, that highlights the limitations of the classical representation in tracking implicit effects. In this example domain, a bread loaf (`bread loaf`) and slices of the loaf (`slice_0`, `slice_1`, `slice_2`) are on a countertop (`countertop`). The goal is to move the bread loaf, and implicitly, all its slices, from the countertop to the kitchen table (`kitchentable`). This example illustrates a failure of the classical representation to preserve global semantics of the world, provided by transitive relations, when actions update on only a part of the world. Using the double-pushout method of C-set rewriting, the categorical representation is able to do so.

Handling Structured Knowledge This example presents a few noteworthy semantic features. A planning representation in this domain must be able to encode the following facts, explicitly or implicitly, at different points in time:

(a) the bread slices are part of the bread loaf

(b) the bread loaf is on the countertop

(c) the bread slices are on the countertop

(d) the bread loaf is on the kitchen table

(e) the bread slices are on the kitchen table

In the case of the categorical representation, fact (a) is captured by the morphism is part of : BreadSlices → BreadLoaf in the schema. Fact (b) about the bread loaf being on the countertop is reified through an abstract object called Object. This is done so that the morphism on : Object → Object can represent the general notion of an object being on top of another object, instead of the more specific relation of a bread loaf being on a countertop. Fact (c) is then captured by the composite morphism on ∘ is a ∘ is part of : BreadSlices → Object. In the classical representation, fact (a) is captured by the propositions `partOf(slice_`n`, bread)` for $n = 0, 1, 2$. Fact (b) and (d) are captured by the proposition

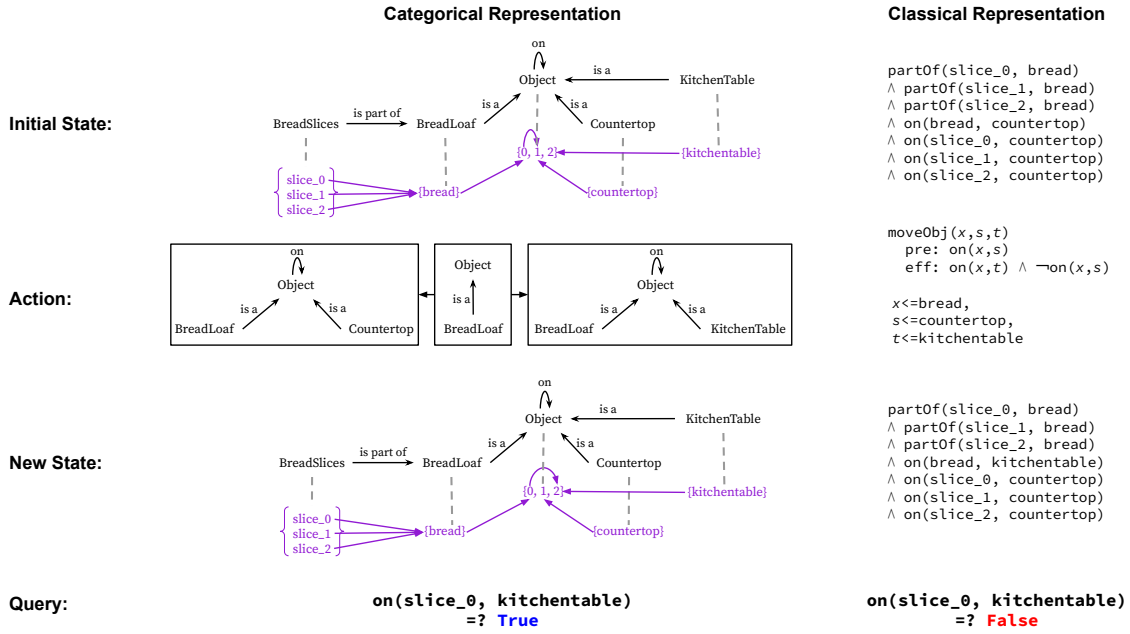


Figure 4.3: A comparison of states, actions, and inferences made between the categorical representation and the classical representation for an example that moves a loaf of bread from a countertop to a kitchen table.

`on(bread, countertop)` and `on(bread, kitchentable)`. Fact (c) and (e) are captured by the propositions `on(slice_`n`, countertop)` and `on(slice_`n`, kitchentable)`. Intuitively, because the bread loaf is on the countertop, the slices that make up the loaf are also on the countertop. In the categorical representation, this captured using a composite morphism. In the classical representation, this is done by explicitly stating for each slice, that it is on the countertop.

Handling Applicability of Actions In the categorical representation, applicability of an action is determined by the existence of a monomorphism in C-Set from the rule input to the world state. Recall that a (covariant) representable functor maps an object, $x \in \mathsf{C}$, to the set of morphisms that have a x as its source. When you present an action using colimits of representables, there are both explicit conditions given by the representables and implicit conditions that appear when the representable is computed. This provides a mechanism for having implicit conditions in rules. In this example, the initial state satisfies the input action rule involving the BreadLoaf, Object, and Countertop. In the classical representation, applicability of the action `moveObj(bread, countertop, kitchentable)` is determined by whether or not the world state contains the element `on(bread, countertop)` in the set of propositions.

Handling Implicit Transitivity The action of moving the bread loaf from the countertop to the kitchen table is applicable in this example. In the categorical representation, this action is modeled by a span in C-Set whose left foot includes knowledge about the bread loaf being on the countertop and whose right foot includes knowledge about the bread loaf being on the kitchen table. The apex of the span states that the bread loaf itself is preserved throughout this change. In the classical representation, the action schema describes the

generic action of moving an objects `x` from one location, `s`, to another, `t`. The action operator is grounded by assigning `x` to the `breadloaf`, `s` to the `countertop`, and `t` to the `kitchentable`. Once this action is applied, a desirable outcome would be for the new state to account for the movement of the bread slices from the countertop to the kitchen table because they are part of the breadloaf. In the categorical representation, the same composite morphism that existed in the initial state exists in the final state; however, the target of the morphism has changed from the countertop to the kitchentable. This captures the implicit change that occurred to the bread slice locations. In the classical representation, the new state captures the fact that the breadloaf is on the kitchen table, but does not capture the fact that the bread slices are on the kitchen table. This is due to the inertia frame axiom, which states that all facts that are not explicitly accounted for in the effect remain true after the action is applied. In practice, this error would likely cause a planner to instruct an agent to move each bread slice individually due to an inconsistency in the world state.

While this example presented a simplied kitchen world, it is easy to imagine a larger domain, such as a manufacturing domain, benefitting from the preservation of global semantics when an action is applied.

4.1.5 Forward Planning with Backtracking using AlgebraicPlanning

Because the components of our framework align with components of the state transition system planning model, it is possible to leverage the long history of work done to design efficient planning algorithms and incorporate heuristics.

Consider that a plan is a sequence of actions that change an initial world state to one that satisfies some goal criteria. To set up a planning problem in our framework, we need to point out two objects in the C-Set that are the initial state and the goal state. We can then consider a plan to be a sequence of rules that when applied to the initial state constructs an object such that a monic map exists from the goal state to that object. We borrow from successful methods in the field of automated planning to implement a forward search algorithm with backtracking [37]. The pseudocode for this implementation can be found in Algorithm 2. The exit criteria involves checking whether a monic map exists from the goal into the current world state.

As with other planning algorithms, this method is subject to issues related to cycles and non-termination. This occurs in scenarios where a rule might be applicable indefinitely if the world specification does not capture a way of destructing an object when a rule is applied. For example, in the sandwich-making example, slicing the bread does not reduce the bread loaf in any manner which means our planner could potentially slice the bread loaf infinitely many times. The integration of attributed C-Set, or "acsets", is in the future of this framework which would allow users to specify attributes for each entity, such as $\mathsf{BreadLoaf} \to \mathsf{NumSlices}$. This structure would provide a well-defined way to do arithmetic and other manipulations with the attributes which could help keep track of resource limits. For now, we use an ad hoc method, a `rule_limit` dictionary that specifies the maximum number of times a rule can be applied in a plan. This implementation will be made available in the AlgebraicJulia ecosystem as part of this thesis. A copy fo the source code can be found in Appendix B.3.

Algorithm 2 Forward Planning with Backtracking

```
1: procedure ForwardPlan(Y world, G goal, r rules, r_usage rule usage, r_limits rule limits, p plan)
2:     if monomorphism G ↪ Y exists then
3:         return Plan p
4:     end if
5:     Initialize applicable rules list, applicable
6:     for rule in r do
7:         Get the input object of rule, r_I
8:         Check if monomorphism r_I ↪ Y exists
9:         if exists then
10:            Append rule to applicable
11:        end if
12:    end for
13:    if applicable is empty then
14:        "No applicable rules!" ThrowException
15:    end if
16:    for a in applicable do
17:        if r_usage[a] ≥ r_limits[a] then
18:            "Rule limit reached!" continue
19:        end if
20:        Y = DPO(Y, representable(a))
21:        Append a to p
22:        ForwardPlan(Y, G, r, r_usage, r_limits, p)
23:    end for
24: end procedure
```

Chapter 5

Ongoing Result - Measuring problem difference

To support our hypothesis, it is necessary that we design a method for evaluating plan stability in light of changes to the problem. In order to this, we would like to provide some measurement of difference between the new planning problem and the old one to ensure that we generate a fair sample of problem changes for each domain. We believe that the formalism outlined in the previous chapters adds relevant and helpful structure that allow changes in the problem to be rigorously defined. In particular, we claim that the formalism captures informative structure about the difference between world states that a STRIPS comparison cannot provide. This will help develop a measurement for comparing world states with respect to a problem instance and a goal. In this chapter, we provide a sketch of how we expect this measurement might work using a simple Blocksworld example for exposition sake.

5.1 Example: Blocksworld

Let us imagine we are in a simple planning domain, such as the Blocksworld domain. We begin with three blocks `A, B, C` where block `A` is on the table and block B is on top of `C`. The goal is to arrange the blocks such that `B` is on top of `A` which is on top of `C`. A drawing is shown in Figure 5.1. The initial world state might look like the following in the language of STRIPS:

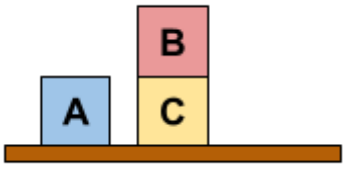


Figure 5.1: Initial Blocksworld problem

```
ontable(A) AND on(B, C) AND ontable(C)
```

To achieve our goal, we would expect a planner to produce a plan that resembles the following.

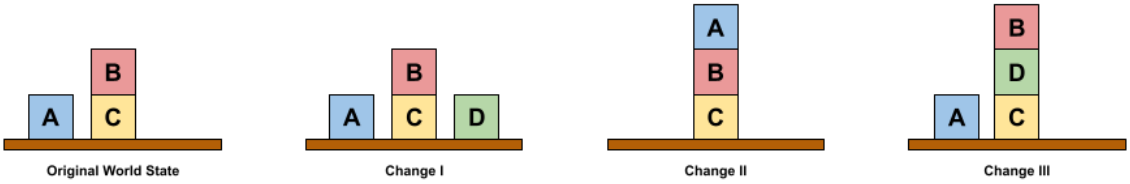


Figure 5.2: Example changes to the Blocksworld problem

```
unstack(B, C)-putdown(B)-pickup(A)-stack(A, C)-pickup(B)-stack(B, A)
```

In this example, we omit the how the actions are specified and assume the standard Blocksworld PDDL domain specification.

5.1.1 Change I - Change number of objects

Now suppose we were to add a block D to the table, while keeping the goal the same. All the changes are drawn in Figure 5.2.

```
ontable(A) AND on(B, C) AND ontable(C) AND ontable(D)
```

You would expect a planner to produce the same plan because block D poses an irrelevant change with respect to the goal.

5.1.2 Change II - Change relations between objects

Now suppose we were to change the arrangement of the initial set of blocks to have Block A on top of B, B on top of C, and C on top of the table:

```
on(A, B) AND on(B, C) AND ontable(C)
```

We would expect that a planner would produce something like:

```
unstack(A, B)-putdown(A)-unstack(B, C)-putdown(B)-pickup(A)-stack(A,
                        C)-pickup(B)-stack(B, A)
```

Notice that the plan is prepended with two additional actions unstack(A)-putdown(A) and the rest of the plan remains the same.

5.1.3 Change III - Change number of objects and change relations

Finally, let us suppose we were to add in Block D and make the arrangement of the blocks to be the following:

```
on(A, B) AND on(B, D) AND on(D, C) AND ontable(C)
```

We would expect that a planner would produce something like:

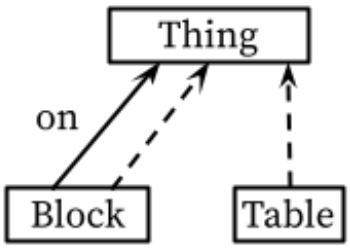


Figure 5.3: Blocksworld schema category, C

```
unstack(A, B)-putdown(A)-unstack(B, C)-putdown(B)-unstack(D,
C)-putdown(D)-pickup(A)-stack(A, C)-pickup(B)-stack(B, A)
```

Part of the original plan was once again prepended with an even longer sequence of actions. All the problem changes are shown in Figure 5.2.

5.1.4 Comparing changes to the world state

If we were to compare all three world state changes, we might like to say that changes between the original problem and the new problem from least to greatest are: Change I < Change II < Change III. This order was informed by the changes we observed after generating a new plan. However, we could also look to the structural dissimiliarity as an indicator for how far we are from the initial state.

So how can we measure this change? If we were to remain in the language of STRIPS and PDDL, we could measure world changes by computing some edit distance between the sets of grounded predicates. This has some qualms because the world changes found in Change I and Change III would be considered equal because they both added a single new ground predicate. This would be coarse assessment of world changes that would not be very useful in more complex settings. Additionally, we could consider a Jaccard similarity for the sets of ground predicates; however, this metric is sensitive to how verbose the problem designer wanted to be when describing the world state. For example, if the problem designer wanted to include all the valid predicates in the Blocksworld vocabulary (on ?x - block ?y - block), (ontable ?x - block), (clear ?x - block), (holding ?x - block) another equally valid initial problem would be:

```
ontable(A) AND on(B, C) AND ontable(C) AND clear(B)
```

The addition of this predicate would alter the score we get for the amount of change we observed in the world, despite being semantically equivalent. We would like a difference measure to be robust to decisions made by a human designer.

If we consider the semantic difference between Change I and Change III, namely the addition of on(D, C) and on(D, table), we will notice that there are explicit and implicit interactions of D with other objects

in the scene. D in Change I is involved in a chain of interactions with A, B, and C; whereas D in Change III interacts solely with the table. Therefore, it would be ideal for a problem difference measure to be sensitive to semantic changes to the world, as opposed to solely syntactic changes.

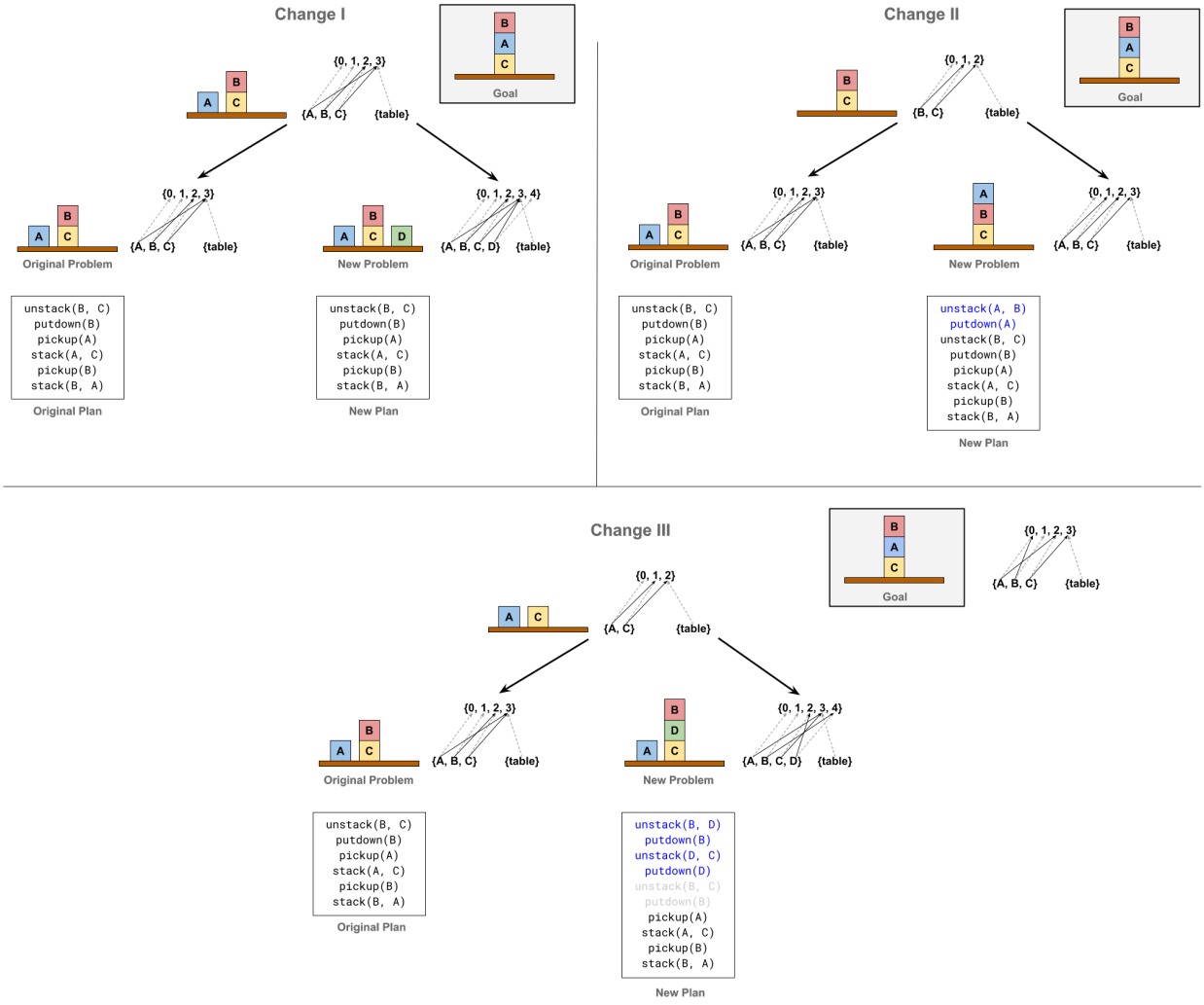


Figure 5.4: Representing Blocksworld problem changes as spans

We propose the use of C-sets as means of expressing world state and, in turn, problem changes. If we were to consider this example using Algebraic Planning, we would express the ontology of our domain using the category, C shown in Figure 5.3. In this category, there are three objects– Block, Table, Thing. There are three morphisms which are shown using the three arrows in the diagram in Figure 5.3. The dotted lines represent is-a relations found in knowledge representation, e.g. (Block, block-is-a, Thing) and (Table, table-is-a, Thing). The only non-taxonomical relation is the one describing the relation of the block being

on athing. Recall from Section 3.1 that C-sets are functors between ontology, C, and semantic category, Set. Therefore, each of these boxes get mapped to sets and each morphism gets mapped to functions. We can express change I through III using AlgebraicPlanning as shown in Figure 5.4.

Each change is represented as a span ($\bullet \leftarrow \bullet \rightarrow \bullet$) of three C-set functors within the category of C-Sets. Within each functor, the gray dotted arrows represent is-a relationships and the black solid arrows are the function mappings from Block to Thing representing the morphism $\mathsf{on}(\mathsf{Block}, \mathsf{Thing})$ in C.

For our prototype difference measure, we can take the Grothendieck construction, or category of elements construction, of a functor F, also noted as $\mathsf{Gr}(\mathsf{F})$. This lifts the functor, F, to a category that contains objects that represent every element in all the sets and morphisms that represent every component of the function map. A more formal definition of the Grothendieck construction can be seen in Definition 2.4.4. For example, some objects in $\mathsf{Gr}(\mathsf{F})$ are (Block, A), (Block, B), (Thing, 2), and (Table, table). This construction also includes composite maps as morphisms in the category. For example, if there existed morphisms $phi : (X, x_1) \rightarrow (Y, y_0)$ and $chi : (Y, y_0) \rightarrow (Z, z_2)$, then the composite morphism $chi \circ phi$ would be included as morphisms in $\mathsf{Gr}(\mathsf{F})$. Because $\mathsf{Gr}(\mathsf{F})$ is also a category, identity morphisms are also included.

As an example, we can take our initial problem to be, F, and we can enumerate the objects and morphisms found in $\mathsf{Gr}(\mathsf{F})$.

> **Objects**: (Block, A), (Block, B), (Block, C), (Thing, 0), (Thing, 1), (Thing, 2), (Thing, 3), (Table, table)
>
> **Morphisms**: (on, A:3), (on, B:2), (on, C:3), (block-is-a, A:0), (block-is-a, B:1), (block-is-a, C:2), (table-is-a, table:3), id_(Block, A), id_(Block, B), id_(Block, C), id_(Thing, 0), id_(Thing, 1), id_(Thing, 2), id_(Thing, 3), id_(Table, table)

In $\mathsf{Gr}(\mathsf{F})$, we have 8 objects and 15 morphisms, among them 7 are non-identity morphisms.

Our prototypical difference measure, $d\delta(F, G)$ shown in Equation 5.1, could be the complement of the Jaccard similarity between the non-identity morphisms of $\mathsf{Gr}(\mathsf{F})$ and $\mathsf{Gr}(\mathsf{G})$, written as $\mathsf{Gr}(\mathsf{F})_1^*$ and $\mathsf{Gr}(\mathsf{G})_1^*$, respectively.

$$\delta(F, G) = 1 - 2(|\mathsf{Gr}(\mathsf{F})_1^* \cap \mathsf{Gr}(\mathsf{G})_1^*|)/(|\mathsf{Gr}(\mathsf{F})_1^*| + |\mathsf{Gr}(\mathsf{G})_1^*|) \tag{5.1}$$

The homomorphism matching algorithm for getting C-set transformations 3.1.1 can be used to identify matches between the two functors. This would allow the intersection of the morphisms between $\mathsf{Gr}(\mathsf{G})$ and $\mathsf{Gr}(\mathsf{F})$ to consider the structural similarity, as opposed to the syntactic one.

This means that for change I, the difference measure is $\delta(F, G) = 1 - 2(7)/(7 + 9) = 0.125$. Based on this metric, we also observe that change II has a difference of 0.286. And change III has a difference of 0.375. The difference between $\delta(F, F)$ is 0.

With this language, there are strict requirements that relationships between objects in the schema are well-defined. When compared with PDDL, a requirement like this is not enforced. For example, it is possible to say that on(A, B) and on(D, B) in PDDL. A situation like this is handled by careful description of the world state or adding axioms that are specific to the Blocksworld domain. For AlgebraicJulia, the metric will be affected proportional to the size of the schema.

5.1.5 Task planning domains

There are many applications of task planning in robotics. In this thesis, we aim to demonstrate this approach on a diverse range of planning domains. Because of this, we have selected the one benchmark planning domain, namely Blocksworld, and two application examples related to manufacturing warehouse automation and home service automation.

Figure 5.5: Candidate domains

5.1.5.1 Blocksworld

The Blocksworld domain is a well-established and widely studied benchmark in the field of artificial intelligence (AI) planning. It is a simplistic, yet representative problem-solving domain that has been used for testing and evaluating various AI planning algorithms and techniques. The domain consists of a finite set of blocks and a table, where the blocks can be stacked on top of each other or placed on the table. The goal is to manipulate the blocks to achieve a specified configuration, subject to certain constraints and rules.

In the Blocksworld domain, the state is represented as a collection of predicates, which describe the relationships between blocks and their positions. Typical predicates used in this domain include `on(X, Y)`, denoting that block X is placed directly on top of block Y, `ontable(X)`, indicating that block X is positioned on the table, and `clear(X)`, meaning that there are no other blocks on top of block X. The objective of a planning problem in the Blocksworld domain is to transform an initial block configuration into a goal configuration.

This domain allows us to evaluate our method against a domain that has limited degrees of freedom. What this means that there is only one parameter that can be changed within the domain which is vary by number of blocks. The number of relations will also be at proportional to the number of blocks because blocks in this world can only interact with one other object, another block or the table.

5.1.5.2 Home Service Automation

In the future, task planning can be instrumental in home service automation, particularly for kitchen task assistance, where robots are required to help with various cooking and cleaning tasks. These scenarios involve complex manipulation, interaction with a variety of objects, and reasoning about recipes or procedures. Task planning can help robots decompose complex kitchen tasks, such as preparing a meal, into smaller, manageable sub-tasks. It can also determine the optimal sequence of actions to perform, considering the dependencies between tasks, cooking times, and resource constraints. Kitchens typically contain a wide range of objects and tools, such as utensils, appliances, and ingredients. Task planning can enable robots to reason about the properties and affordances of these objects, selecting the appropriate tools and actions for each task, and ensuring proper handling and manipulation. Additionally, kitchen environments can be dynamic and uncertain, with changing conditions or unexpected events, such as spills or ingredient shortages.

In recent years, the application of robot task planning for kitchen assistants has received considerable attention in both academic and industry research. Beetz et al. [13] demonstrated the potential of autonomous robots in the kitchen by developing a system capable of preparing pancakes, involving complex manipulation

tasks and reasoning about actions, objects, and dependencies. Similarly, Tenorth and Beetz [77] developed KnowRob, a knowledge processing framework that allows autonomous personal robots to perform various kitchen tasks, such as sandwich-making, by incorporating knowledge representation and reasoning techniques. In the context of home service and kitchen automation, numerous studies have focused on developing robotic systems that can handle a wide range of kitchen tasks, such as meal preparation, setting the table, and dishwashing [23]. However, the successful integration of task planning in kitchen assistants remains an ongoing challenge, as it involves addressing issues such as object recognition, manipulation, and planning actions with temporal and spatial constraints. Nevertheless, advances in robotic manipulation, planning algorithms, and human-robot interaction continue to pave the way for more capable and efficient kitchen assistant robots in the near future. We feel this is an appropriate domain to evaluate our framework on a "medium-sized" problem. It is also a friendly domain to present to non-experts in robotics and task planning.

The Sandwich-Making Problem In particular, we will focus on the sandwich-making problem in order to demonstrate the challenges and opportunities that arise when applying task planning techniques to kitchen settings. In this problem, a robot must autonomously perform a series of actions to prepare a sandwich, including selecting ingredients, slicing bread, spreading condiments, and assembling layers, all while adhering to specific spatial constraints and goals. Kitchen environments present dynamic and uncertain scenarios, requiring robots to generate stable plans that are robust to changing circumstances, such as ingredient availability, preferences, or unexpected obstacles.

5.1.5.3 Manufacturing and assembly

Task planning offers a high-level alternative to traditional robotic programming methods, enabling more abstract, flexible, and scalable representations of tasks and environments. Traditional robotic programming often involves explicitly defining sequences of low-level actions, control policies, or behaviors to achieve specific goals. This approach can be time-consuming, inflexible, and hard to adapt to new tasks or environments. In contrast, task planning focuses on reasoning about the world using high-level representations, making it a versatile and adaptable approach to robotic programming.

Today, industrial programming methods include lead-through programming and off-line programming. Lead-through programming remains the most widespread approach to robot programming in industry. In the case of lead-through programming, the human worker manually manipulates the robot end-effector to a desired position and saves that position to its trajectory using a teach pendant [83]. Each position may specify additional signals that trigger actions such as gripping or welding. This is similar to a stop motion animator moving objects and capturing the frame. As in the case of stop-motion animation, this is straightforward and intuitive; however, it is also very time-consuming and requires repeating this exercise for every new or modified task or work cell configuration [85]. This introduces downtime for reprogramming and requires specialized expertise to program efficient and cooperative trajectories.

To reduce downtime, off-line programming methods were introduced [83]. These methods require a high-fidelity digital model of the real-world robot and work cell, and a procedural program that specifies *how* the task must be accomplished [59]. Off-line programming languages are flexible, allow processing of sensor inputs, and allow implementation of advanced perception algorithms and simulations. They, however, demand that a roboticist be capable of developing complex computer programs, making this method a less accessible option. In practice, most robot-level programs are point-solutions that work on specific robot platforms, sensors, and algorithms.

To mitigate this, a programming method called goal-oriented robot programming (GoP), also referred to as task-level programming is used to describe the goal in terms of operations on the objects participating

in the task [59]. This allows tasks to be expressed functionally, in terms of *what* must be accomplished. GoP systems require access to a knowledge base consisting of geometric models of their environment and the robot, such as the digital model created in off-line programming. This knowledge base is often referred to as a *world model*. World models can be built with computer-aided design (CAD) models (that contain mass, inertia, kinematic descriptions, joint limits) and simulation engines (that generate dynamics), and updated from sensors such as cameras to provide updates from the real world. GoP systems provide a programming framework to compose programs using motion primitives, skills, and tasks [14]. GoP systems require a compiler to transform task-level specifications into robot-level specifications. The compiler's function is to generate a robot-level program that manipulates objects in the environment from an initial state to a desired final state. GoP provides the task-level specifications and places responsibility on the roboticist to write a goal-oriented program to achieve the goal. Task planning could potentially be used in place of the roboticist to determine the correct program. We feel this is an appropriate domain to evaluate our framework on a large problem. Programming robotics to accomplish tasks often require a considerable amount of physical and spatial information represented in the world model, making it a suitable domain for evaluating how large of a domain our approach can empirically handle.

The Machine-Tending Problem In particular, task planning can play a vital role in automating and optimizing manufacturing processes, such as machine-tending applications. In such scenarios, robots are responsible for loading and unloading materials, workpieces, or tools to and from machines, ensuring smooth operation, efficiency, and productivity in the manufacturing process. For example, task planning can aid robots in identifying the optimal sequence of loading and unloading operations across multiple machines, considering the constraints and objectives of the production process. This would ensure efficient resource allocation, minimize idle time, and maximize throughput. Additionally, in real-world manufacturing settings, the environment is often dynamic, with unexpected events such as machine breakdowns, changes in production orders, or resource constraints [25]. In theory, task planning would be robust to these changes. However, in practice, this is rarely the case when using classical approaches due to underspecified domain models. This makes it difficult to accurately assess the impact of a change to the plan [22].

5.1.5.4 Comparing machine-tending with sandwich-making

There are many similarities and differences between the sandwich-making problem and the machine-tending problem. The major difference is the scale at which objects in the kitchen domain vary. For example, in the kitchen example, within a given ingredient, there may be many valid candidates for that ingredient; therefore, reasoning about substitutions is critical to the success of reasoning in this domain. Additionally, sandwich ingredients and tools necessary vary dramatically based on the consumer's preferences. This contrasts the machine-tending example because the type of materials and methods for part preparation are standardized and thoroughly defined. In the machine-tending scenario, it is more likely that the attributes of a part are more abundant than in ingredients in the sandwich-making scenario. Being able to reason about both scenarios using the same representation will demonstrate the power of this approach in handling large varieties of objects, classes, and constraints; such as substitutions and descriptive attributes.

Chapter 6

Proposed Work and Timeline

This thesis, ultimately, aims to address of the problem of plan adaptation in light of changes to a complex world state.

6.1 Summary of Proposed Work

Below is a high-level summary of the research we presented in this prospectus.

Preliminary Result I - C-set for representing complex world states In this work, we propose a novel use of the formalism for typed graphs, called C-set which is capable of expressing ontologies and instances with functional and compositional properties. In particular, we demonstrate its usefulness in encoding world states as scene graphs in a principled manner. This formalism does so by establishing a functorial relationship between a syntax category and a semantics category, called functorial semantics, which induces compositional or path equivalence relations between classes and their corresponding instances. This is beneficial for (i) managing object, relations, and attributes present in domains with a rich typeclass hierarchy, (ii) ensuring world states adhere to the prescribed structure, namely the ontology, unambiguously, and (iii) handling implicit transitive relations induced by composable paths in the ontology. For additional detail, see Chapter 3.

Preliminary Result II - Planning using C-set and DPO rewriting In this work, we propose the use of C-set for representing world states in AI task planning. Task planning is responsible for identifying a sequence of high-level actions that translate an initial state to a goal state. Some problems in the field include difficulty managing large and complex world states and preserving transitive closure. We claim that these problems arise due to the limitations of existing representation languages, namely as PDDL and STRIPS. Towards a defense, we demonstrate that the proposed formalism is capable of addressing these issues through an example. For additional detail, see Chapter 4.

Ongoing Result - Measuring problem difference In this work, we discuss how our proposed formalism is provides us with the structure for measuring changes in the problem. We will outlined the three domains for which of category theoretic structures for task planning can be applied. We propose to demonstrate the usability and benefits of employing functorial semantics to AI task planning in a range of case studies,

namely, a canonical planning task, machine-tending in manufacturing, and sandwich-making in home service automation.

6.2 Evaluation

Recall that the, parts of this work aim to support the following hypothesis.

Hypothesis A. We hypothesize that by developing a planning framework that effectively addresses the challenges of:

(i) representing complex and structured world states,

(ii) reasoning about transitive relations, and

(iii) remaining interoperable with other AI components for neurosymbolic reasoning,

we will be able to generate higher quality plans and enhance the applicability of task planning in complex real-world domains relative to existing planners.

In order to support this hypothesis, we have identified a few hypotheses for which we will explain how the each is or will be supported.

6.2.1 Evaluating subordinate hypotheses

Hypothesis A.1 In support of (i), we claim that all scene graphs can be represented as typed graphs.

This will be supported by providing a proof showing that there is a bijective correspondence (a one-to-one mapping) between the elements of scene graphs, expressed using the RDF standard [84], and the elements of typed graphs. We will discuss what restrictions will need to be applied to RDF, if necessary.

Hypothesis A.2 In support of (ii), we claim that C-sets and DPO rewriting preserves transitive relations expressed in the ontology after every action.

This is supported by the definition of a functor and a natural transformation. See Section 4.

Hypothesis A.3 In support of (iii), we claim that arbitrary scene graphs based on a large number of entities and relations can be translated into C-sets in polynomial time.

This will be supported by designing an algorithm that that takes an abitrary scene graph expressed using the RDF triples [84] and converts it to a valid C-set, evaluate the complexity of the algorithm based on its steps, and provide an implementation of the algorithm in AlgebraicPlanning. Empirical results of the conversion will also be obtained.

Hypothesis A.4 In support of higher quality plans, we claim that the proposed formalism produces more concise plans (fewer steps) at an equal or higher success rate than an existing general-purpose planner, Metric-FF [42], that considers typing with and without transitive closure domain axioms for a range of domains (those with at least k-entities and m-relations in their ontology, where $k = [5, 10, 100]$, for now, and m will be determined) and problem sizes (those with n-objects in the domain, where

$n = [5 : 1000]$ will depend on the specific domain).

Prior to running the experiment, we will implement a Julia package, called AlgebraicPlanning, with the following components.

(a) A description and implementation of a C-set based planning language based on Section 3.1

(b) An interface for defining planning problems and domains based on C and rewrite rules.

(c) A planning algorithm that conducts forward planning with backtracking planning based on the proposed formalism based on Section 4.1.3

(d) A measurement for assessing the difference between C-sets basd on Section 5.1.4

(e) A tool for generating problems as C-sets of varying size provided a domain ontology and a range of instances.

Provided these items, we will run the following experiment whose setup is largely based on the canonical experiment set-up by Howe and Dahlman [45].:

1. Select and construct planning domains based on Blocksworld, Sandwich-making, and Machine-tending domains described in Section 5.1.5 in both our formalism and in PDDL.
2. For each domain, construct a problem set consisting of p problems that is uniformly distributed according to the proposed measurement. At this time, a reasonable $p = 100$; however, during experimentation, more or fewer test cases might be more feasible or significant.
3. Set-up MetricFF with default parameters and, for each domain, set-up MetricFF to handle derived predicates related to transitive closure, which we will call MetricFF-Transitive. We anticipate this will be a time-consuming process.
4. Run (a) our proposed planning, (b) MetricFF, and (c) MetricFF-Transitive against the problem set, setting an upper limit of 30 minutes to obtain a solution.
5. Record which problems were solved, how many plan steps were in the solution, how much CPU time and memory was required, and any failures or time outs.
6. Analyze and discuss the results against the hypothesis.

Hypothesis A.5 In support of higher quality plans, we claim that the proposed formalism produces more stable plans than an existing general-purpose planner, Metric-FF [42], that considers typing with and without transitive closure domain axioms for a range of domains (those with at least k-entities and m-relations in their ontology, where $k = [5, 10, 100]$, for now, and m will be determined) and problem sizes (those with n-objects in the domain, where $n = [5 : 1000]$ will depend on the specific domain).

To support this hypothesis, we will run a similar experiment except evaluate for plan stability based on a fix problem.

1. Select and construct planning domains based on Blocksworld, Sandwich-making, and Machine-tending domains described in Section 5.1.5 in both our formalism and in PDDL.

2. **(NEW)** For each domain, identify a problem that will produce the baseline plan. This problem will be based on a realistic scenario to the best of our ability.

3. For each domain, construct a problem set consisting of p problems that is uniformly distributed according to the proposed measurement. At this time, a reasonable $p = 100$; however, during experimentation, more or fewer test cases might be more feasible or significant.

4. Set-up MetricFF with default parameters and, for each domain, set-up MetricFF to handle derived predicates related to transitive closure, which we will called MetricFF-Transitive. We anticipate this will be a time-consuming process.

5. **(NEW)** Run (a) our proposed planning, (b) MetricFF, and (c) MetricFF-Transitive *against the baseline problem*, setting an upper limit of 30 minutes to obtain a solution to produce π_0.

6. Run (a) our proposed planning, (b) MetricFF, and (c) MetricFF-Transitive against the problem set, setting an upper limit of 30 minutes to obtain a solution to produce π_i where $i = 1, ..., p$.

7. Record the plan stability between π_0 and π_i.

8. Analyze and discuss the results against the hypothesis. We will also highlight any noteworthy, domain-specific observations, such as interesting scenarios that produced a plan stability of $D(\pi_0, \pi_j) = 0$, which implies no change to the plan.

6.3 Dissertation Plan

Phase	Paper	Tasks	Deadline
Use formalism for planning	ICAPS'24	• Implement prototype planner using formalism in AlgebraicPlanning • Implement and analyze algorithm for converting RDF to C-sets in support of Hypothesis **A.1** and **A.3** • Implement interfaces for defining planning domain and problem • Release as open-source package	OCT 2023
Build problem generator	Artificial Intell., Robotics and Autonomous Sys., IEEE-TASE	• Implement method for measuring difference between problems • Implement scenario generator and sampler that captures world state changes	JAN 2024
Run experiments	AAAI	• Implement test framework with desired performance metrics for plan quality • Set-up MetricFF planner with appropriate configurations (with and without derived predicates) • Run experiments in support of **Hypothesis A.4** and **A.5**	JUN 2024
PhD Defense	UMD	• Write PhD thesis	SEPT 2024

6.4 Paper Goals

The following paper deadlines outline my planned goals for publication. Each publication represents a milestone in my research and an end point for that particular line of investigation. For instance, the ICAPS 2024 paper should mark the end of my research in implementing the AlgebraicPlanning planner.

- *"AlgebraicPlanning: A category theory-based, computational system for symbolic task planning"*
  - ICAPS'24, due November 2023
- *"What difference does it make? Generating world states as typed graphs based on desired degree of change"*
  - IEEE Robotics and Automation, rolling deadline
  - Robotics and Autonomous Systems, rolling deadline
  - IEEE T-ASE, rolling deadline
  - Artificial Intelligence, rolling deadline
- *"The power of change: A framework for assessing the impact of world changes on plan stability"*
  - AAAI'24, due September 2024
  - Artificial Intelligence, rolling deadline
  - Robotics and Autonomous Systems, rolling deadline

Chapter 7

Conclusion

In conclusion, this work proposes a novel approach to the task of AI planning within complex environments. By focusing on the development of a robust representation language, AlgebraicPlanning, based on category theory and leveraging ontology-based typed graphs, we aim to better capture and reason about complex world states and the relationships between objects within them. We tackle three central problems plaguing current AI planning: the difficulty in expressing complex and structured world states, the struggle with reasoning about transitive relations, and the challenge of integrating task planning with other AI components to support neurosymbolic reasoning. Addressing these challenges, we believe, will enable us to generate higher quality plans and enhance the applicability of task planning in complex real-world domains.

To validate our approach, we have proposed several hypotheses, and we will assess the performance of our approach against these. We plan to conduct a series of case studies that explore planning in diverse domains, such as classical planning, manufacturing, and home service robotics, to gauge the breadth and depth of our approach's applicability. Furthermore, we aim to contribute a method for plan quality assessment, providing a systematic way to examine plan stability relative to changes in the world, and giving insight into the relevance of features in the world to achieving a goal.

In the larger scope, this work will provide a concrete example of the application of category theoretic formalisms to AI task planning, enriching the field with a new perspective and potentially paving the way for future research on the intersection of these domains. Given the complexity and dynamic nature of real-world environments, the advancement of AI task planning is crucial for the future of intelligent systems. As such, the contributions from this work will not only have academic significance but also practical implications for a range of industries and applications.

Appendix A

Example PDDL Problems and Domains

A.1 Blocksworld

Here is a simple PDDL domain and problem file for the Blocksworld scenario.

A.1.1 Domain

```
(define (domain blocksworld)
    (:requirements :strips :typing)
    (:predicates
      (on ?x - object ?y - object)
      (ontable ?x - object)
      (clear ?x - object)
      (hand-empty)
      (holding ?x - object)
    )
    (:action pick
      :parameters (?x - object)
      :precondition (and (ontable ?x) (clear ?x) (hand-empty))
      :effect (and (holding ?x) (not (ontable ?x)) (not (hand-empty)))
    )
    (:action put
      :parameters (?x - object)
      :precondition (holding ?x)
      :effect (and (ontable ?x) (clear ?x) (not (holding ?x)) (hand-empty)
          )
    )
    (:action stack
      :parameters (?x - object ?y - object)
      :precondition (and (holding ?x) (clear ?y))
      :effect (and (on ?x ?y) (not (clear ?y)) (not (holding ?x)) (hand-
          empty))
    )
```

```
    (:action unstack
      :parameters (?x - object ?y - object)
      :precondition (and (on ?x ?y) (clear ?x) (hand-empty))
      :effect (and (holding ?x) (clear ?y) (not (on ?x ?y)) (not (hand-
          empty)))
    )
  )
```

A.1.2 Problem

```
(define (problem blocksworld-problem)
  (:domain blocksworld)
  (:objects
    A B C - object
  )
  (:init
    (ontable A)
    (ontable B)
    (ontable C)
    (clear A)
    (clear B)
    (clear C)
    (empty-hand)
  )
  (:goal
    (and (on A B) (on B C))
  )
)
```

Appendix B

AlgebraicPlanning.jl Implementation

B.1 Sandwich-Making Example

B.1.1 Domain

This ontology and rules are expressed using the interfaces provided by Catlab.jl.

B.1.1.1 Ontology

In this simple ontology of a kitchen, there are three main entities: `Entity`, `Food`, and `Kitchenware`. `Food` can be an `Entity` or it can be in/on an `Entity` Similarly, `Kitchenware` can be an `Entity` or it can be in/on an `Entity`. This specification assigns the set {`:Counter, :Fridge, Stove`} to `Entity`, the set {`:Breadloaf, BreadSlice, EggCarton, Egg, CookedEgg, YolkWhite, CheeseBag, Cheese, Mayo, MayoJar, Tomato, Lettuce, Bacon, CookedBacon`} to `Food`, and the set {`:Bowl, :Knife, :Plate, :PaperTowel, :Skillet`} to `Kitchenware`. The `:` is just the prefix to indicate that it is a Symbol type in Julia.

```
module World
export SpecKitchen, SpecBreakfastKitchen, SchBreakfastKitchen, BreakfastKitchen,
    yBreakfastKitchen

using Catlab, Catlab.Theories, Catlab.CategoricalAlgebra
using AlgebraicPlanning

@present SpecKitchen(FreeMCategory) begin
    Entity::Ob

    Food::Ob
    food_in_on::Hom(Food, Entity)
    food_is_entity::Hom(Food, Entity)
    ::Tight(food_is_entity)

    Kitchenware::Ob
    ware_in_on::Hom(Kitchenware, Entity)
    ware_is_entity::Hom(Kitchenware, Entity)
    ::Tight(ware_is_entity)
end

const SpecBreakfastKitchen = copy(SpecKitchen)

```

```
add_entity!.(Ref(SpecBreakfastKitchen), [
  :Counter,
  :Fridge,
  :Stove,
])

add_food!.(Ref(SpecBreakfastKitchen), [
  :BreadLoaf,
  :BreadSlice,
  :EggCarton,
  :Egg,
  :CookedEgg,
  :YolkWhite,
  :CheeseBag,
  :Cheese,
  :Mayo,
  :MayoJar,
  :Tomato,
  :Lettuce,
  :Bacon,
  :CookedBacon
])

add_kitchenware!.(Ref(SpecBreakfastKitchen), [
  :Bowl,
  :Knife,
  :Plate,
  :PaperTowel,
  :Skillet
])
```

Listing B.1: Breakfast Sandwich-Making Domain

B.1.1.2 Rules

Here rules, or action models, are defined using spans of C-sets. The keys `I, O, K` mirror those discussed in Chapter 4. Each line within the macro (`@join`) contains an instance and a type mapping. These are generic mappings, i.e. `loaf::BreadLoaf` refers to some `loaf` and not a specific loaf in the instance of the world. This could have easily had said `foo::BreadLoaf`. The homomorphism matching algorithm takes care of finding the loaf in the world that can serve as a suitable match.

```
const action_rule_schemas = Dict(

:slice_bread => @migration(SchDB, begin
    I => @join begin
      loaf::BreadLoaf
      knife::Knife
    end
    O => @join begin
      loaf::BreadLoaf
      slice::BreadSlice
      food_in_on(bread_slice_is_food(slice)) == food_in_on(bread_loaf_is_food(loaf))
      knife::Knife
    end
    K => @join begin
      loaf::BreadLoaf
```

```
      knife::Knife
    end
end),

:put_cheese_on_bread => @migration(SchDB, begin
    I => @join begin
      bag::CheeseBag
      slice::BreadSlice
    end
    O => @join begin
      bag::CheeseBag
      cheese::Cheese
      slice::BreadSlice
      cheese_is_food(cheese) == bread_slice_is_food(slice)
    end
    K => @join begin
      bag::CheeseBag
      slice::BreadSlice
    end
end),

:crack_egg_in_bowl => @migration(SchRule, SchDB, begin
    I => @join begin
      egg::Egg
      bowl::Bowl
    end
    O => @join begin
      yolk_white::YolkWhite
      bowl::Bowl
      food_in_on(yolk_white_is_food(yolk_white)) == ware_is_entity(bowl_is_ware(bowl))
      orig_container::Entity
    end
    K => @join begin
      egg_entity::Entity
      bowl::Bowl
      orig_container::Entity
    end
    i => begin
      egg_entity => food_is_entity(egg_is_food(egg))
      bowl => bowl
      orig_container => food_in_on(egg_is_food(egg))
    end
    o => begin
      egg_entity => food_is_entity(yolk_white_is_food(yolk_white))
      bowl => bowl
      orig_container => orig_container
    end
end),

:put_skillet_on_stove => @migration(SchRule, SchDB, begin
    I => @join begin
      skillet::Skillet
      stove::Stove
    end
    O => @join begin
      skillet::Skillet
      stove::Stove
      orig_surface::Entity
```

```
      ware_in_on(skillet_is_ware(skillet)) == stove_is_entity(stove)
    end
    K => @join begin
      skillet::Skillet
      stove::Stove
      orig_surface::Entity
    end
    i => begin
      skillet => skillet
      stove => stove
      orig_surface => ware_in_on(skillet_is_ware(skillet))
    end
    o => begin
      skillet => skillet
      stove => stove
      orig_surface => orig_surface
    end
end),

:put_egg_in_skillet => @migration(SchDB, begin
    I => @join begin
      yolk_white::YolkWhite
      bowl::Bowl
      skillet::Skillet
      food_in_on(yolk_white_is_food(yolk_white)) == ware_is_entity(bowl_is_ware(bowl))
    end
    O => @join begin
      yolk_white::YolkWhite
      bowl::Bowl
      skillet::Skillet
      food_in_on(yolk_white_is_food(yolk_white)) == ware_is_entity(skillet_is_ware(skillet))
    end
    K => @join begin
      bowl::Bowl
      skillet::Skillet
    end
end),

:cook_egg => @migration(SchDB, begin
    I => @join begin
      yolk_white::YolkWhite
      skillet::Skillet
      food_in_on(yolk_white_is_food(yolk_white)) == ware_is_entity(skillet_is_ware(skillet))
    end
    O => @join begin
      cooked_egg::CookedEgg
      skillet::Skillet
      food_in_on(cooked_egg_is_food(cooked_egg)) == ware_is_entity(skillet_is_ware(skillet))
    end
    K => @join begin
      skillet::Skillet
    end
end),

:put_egg_on_sandwich => @migration(SchDB, begin
    I => @join begin
      egg::CookedEgg
      skillet::Skillet
```

```
      slice::BreadSlice
      food_in_on(cooked_egg_is_food(egg)) == ware_is_entity(skillet_is_ware(skillet))
    end
    O => @join begin
      egg::CookedEgg
      skillet::Skillet
      slice::BreadSlice
      cooked_egg_is_food(egg) == bread_slice_is_food(slice)
    end
    K => @join begin
      skillet::Skillet
      slice::BreadSlice
    end
end)
)
```

Listing B.2: Breakfast Sandwich-Making Initial State

B.1.2 Problem

The problem is defined by defining a C-set functor for the initial state and the goal state.

B.1.2.1 Initial State

The initial state in this problem states that many objects existing in the world, such as a counter, a fridge, a bread, a knife, etc. with no interactions between them. Under the hood, however, each object is related to the `Entity` class and therefore there exists mapping from each object to distinct entities assigned to type `Entity`.

```
using Catlab, Catlab.CategoricalAlgebra, Catlab.Programs, Catlab.Graphics
using AlgebraicPlanning

world_diagram = @free_diagram World.SchBreakfastKitchen begin
    counter::Counter
    fridge::Fridge
    bread::BreadLoaf
    knife::Knife
    cheesebag::CheeseBag
    egg::Egg
    skillet::Skillet
    bowl::Bowl
  end;
```

Listing B.3: Breakfast Sandwich-Making Initial State

B.1.2.2 Goal State

The goal state states that there exists a bread slice, cheese, and a cooked egg, for which the cooked egg maps to the same food as the bread slice indicating a notion of merging or fusing these objects. The same is said for the cheese and the bread slice.

```
using Catlab, Catlab.CategoricalAlgebra, Catlab.Programs, Catlab.Graphics
using AlgebraicPlanning
world_diagram = @free_diagram World.SchBreakfastKitchen begin
```

```
    slice::BreadSlice
    cheese::Cheese
    egg::CookedEgg

    cooked_egg_is_food(egg) == bread_slice_is_food(slice)
    cheese_is_food(cheese) == bread_slice_is_food(slice)
  end;
```

Listing B.4: Breakfast Sandwich-Making Goal State

B.2 Plan

The FF.jl planner, shown in Section B.3, produced the following plan given the domain and problem.

```
crack_egg_in_bowl
put_egg_in_skillet
cook_egg
slice_bread
put_cheese_on_bread
put_egg_on_sandwich
```

Listing B.5: Breakfast Sandwich-Making Plan

B.3 FF.jl

```
export FF

using Catlab, Catlab.CategoricalAlgebra, Catlab.Programs, Catlab.Graphics, Catlab.Theories
using Catlab.Graphics.Graphviz: run_graphviz
using Catlab.CategoricalAlgebra.DataMigrations:
  DeltaSchemaMigration, ConjSchemaMigration

""" mutable struct FFConfig

"""
mutable struct FFConfig
  plan::Vector{Any}
  num_step::Int
  rule_usage::Dict
end

""" FFConfig(plan::Vector{Any}=[], num_step::Int=0, rule_usage::Dict=Dict())
FFConfig accepts optional parameters
- (`plan::Vector{Any}=[]`) an initial plan which will preppend the found plan. Default is an
    empty vector.
- (`num_step::Int=0`) an integer specifying the depth within the search tree. This is zero-
    indexed.
- (`rule_usage::Dict=Dict()`) a dictionary tracking the number of times a rule has been
    applied in the current branch. If backtracking occurs, this dictionary will revert back
    to the state at the backtracked node in the tree.
"""
function FFConfig(plan::Vector{Any}=[], num_step::Int=0, rule_usage::Dict=Dict())
  return FFConfig(plan, num_step, rule_usage)
end
```

```
26
27 """ FF(world::ACSet, goal::ACSet, rules_dict::Dict, rule_limits::Dict, config::FFConfig)
28 Executes forward planning search to find a sequence or all sequences of applicable DPO
       rewriting rules. A sequence of applicable DPO rewriting rules that transform an initial
       state to a goal state is called a _plan_.
29
30 This is a recursive function that takes a current state (world) and goal state (goal) in the
        form of ACSets, a dictionary of rules, and dictionary of rule limits. Rule limits are
       used to limit the number times a given rule is applied with a plan. This is necessary to
        prevent looping within a branch.
31 """
32 function FF(world::ACSet, goal::ACSet, rules_dict::Dict, rule_limits::Dict, config::FFConfig
       )
33
34   # Exit criteria: goal is reached
35   if homomorphism(goal, world, monic=true) !== nothing
36     println("Goal reached.")
37     return plan
38   end
39
40   applicable = find_applicable_rules(world, rules_dict)
41
42   # Backtrack criteria: no applicable rules
43   if length(applicable) <= 0
44     throw("No applicable rules found! Terminate path...")
45   end
46
47   try
48     for rule_hom in applicable  # try all applicable rules in order
49       selected = rule_hom.first
50
51       # Backtrack criteria: rule applied too many times
52       try
53         if config.rule_usage[selected] >= rule_limits[selected]
54           println("Abort path. Rule used too many times.")
55           continue
56         end
57       catch
58         config.rule_usage[selected] = 1  # fails if key doesn't exist, initialize
59       end
60
61       # apply rule and add to plan
62       prev_world = world
63       try
64         world = rewrite(rules_dict[selected].rule, world)
65       catch e
66         throw("Could not apply rule! " * string(e))
67       end
68       append!(config.plan, [rules_dict[selected].rule => rule_hom.second])  # include both
       rule and matched homomorphism
69
70       try
71         config.num_step =+ 1
72         config.plan = FF(world, goal, rules_dict, rule_limits, config)
73         return config.plan
74       catch e
75         world = prev_world
76         pop!(config.plan)
```

```
77          println("Path failed. Try a different path.")
78          # println(string(e))
79        end
80      end
81    catch e
82      throw("Something went wrong with error: " * string(e))
83    end
84
85    println("No plan found.")
86  end
```

Listing B.6: FF.jl

Appendix C

Past papers and presentations

C.1 Papers

1. Aguinaldo A., Patterson E., Fairbanks, J., Ruiz J. A Categorical Representation Language and Computational System for Knowledge-Based Planning. Under review. 2023.

2. Aguinaldo A., Regli W. Modeling traceability, change information, and synthesis in autonomous system design using symmetric delta lenses. ICRA Compositional Robotics Workshop 2022.

3. Aguinaldo A., Regli W. Encoding Compositionality in Classical Planning Solutions. IJCAI Workshop on Generalization in Planning 2021.

4. Aguinaldo A., Regli W. A Graphical Model-Based Representation for Classical AI Plans using Category Theory. ICAPS 2021 Workshop on Explainable AI Planning.

5. Aguinaldo A., Bunker J., Pollard B., Shukla A., Canedo A., Quiros G., Regli W. RoboCat: A Category Theoretic Framework for Robotic Interoperability Using Goal-Oriented Programming. IEEE Transactions on Automation Science and Engineering, 2021. doi: 10.1109/TASE.2021.3094055.

C.2 Presentations

1. *"A Category Theoretic Approach to Planning in a Complex World"* at Microsoft Future Leaders in Robotics and AI Seminar Series in 2023

2. *"Contextual affordances in context-aware autonomous systems"* at AMS JMM 2023 Special Session on Applied Category Theory (a Mathematics Research Communities session)

3. *"Contextual affordances in context-aware autonomous systems"* at NIST Compositional Structures for Systems Engineering and Design Workshop 2022

4. *"Category theory for automated planning and program compilation in robotics"* at Topos Berkeley Seminar 2022

5. *"Applications of category theory to automated planning and program compilation in robotics"* at Xerox PARC Design Seminar in 2022

6. *"Modeling traceability, change information, and synthesis in autonomous system design using symmetric delta lenses"* at International Conference for Robotics and Automation (ICRA) 2022 Compositional Robotics Workshop in 2022
7. *"Encoding Compositionality in Classical Planning Solutions"* at International Joint Conference for Artifical Intelligence (IJCAI): Workshop on Generalization in Planning in 2021
8. *"A Graphical Model-Based Representation for Classical AI Plans using Category Theory"* at International Conference for Automated Planning and Scheduling (ICAPS): Explainable AI Planning Workshop in 2021
9. *"Polynomial Functors in Catlab"* at the Applied Category Theory Conference in 2021
10. *"Diary of a software engineering using categories"* at Topos Institute: Berkeley Seminar and Lógicos em Quarentena (LQ) Seminar in 2021
11. *"Category Theory for Software Modeling and Design"* at Hunter College Applied Mathematics Seminar in 2020

C.3 Extracurricular

- Open-source contributor to AlgebraicJulia
- Co-organizer of Compositional Robotics: Mathematics and Tools (ICRA 2023 Workshop), 2023
- Co-organizer of The Adjoint School, 2022-2023
- Student of The Adjoint School, 2021

Appendix D

Reading List

D.1 Area 1 – Knowledge Representation

This reading list focuses on typed graphs, scene graphs, and knowledge graphs in robotics.

1. Galindo, C., Fernández-Madrigal, J. A., González, J., & Saffiotti, A. (2008). Robot task planning using semantic maps. Robotics and Autonomous Systems, 56(11), 955–966. https://doi.org/10.1016/j.robot.2008.08.007

2. Tenorth, M., & Beetz, M. (2017). Representations for robot knowledge in the KNOWROB framework. Artificial Intelligence, 247, 151–169. https://doi.org/10.1016/j.artint.2015.05.010

3. Miao, R., Jia, Q., & Sun, F. (2023). Long-term robot manipulation task planning with scene graph and semantic knowledge. Robotic Intelligence and Automation, November. https://doi.org/10.1108/ria-09-2022-0226

4. Agia, C., Jatavallabhula, K. M., Khodeir, M., Miksik, O., Vineet, V., Mukadam, M., Paull, L., & Shkurti, F. (2021). TASKOGRAPHY: Evaluating robot task planning over large 3D scene graphs. 5th Conference on Robot Learning, CoRL, 1–18. http://arxiv.org/abs/2207.05006

5. Armeni, I., He, J. Z., Gwak, J., Zamir, A. R., Fischer, M., Malik, J., & Savarese, S. (2019). 3D Scene Graph. Iccv. http://3dscenegraph.stanford.edu

D.2 Area 2 – AI Planning

This reading list focuses on typed planning and derived predicates.

1. Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, 20, 61–124. https://doi.org/10.1613/jair.1129

2. Hoffmann, J. (2003). The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. Journal of Artificial Intelligence Research. https://doi.org/10.1613/jair.1144

3. Thiebaux, S., Hoffmann, J., & Nebel, B. (2000). In Defense of PDDL Axioms. Syntax And Semantics, 1–6.

4. Howe, A. E., & Dahlman, E. (2002). A critical assessment of benchmark comparison in planning. Journal of Artificial Intelligence Research, 17, 1–33. https://doi.org/10.1613/jair.935

5. Fox, M., Gerevini, G., Long, D., & Serina, I. (2006). Plan stability: Replanning versus plan repair. ICAPS 2006 - Proceedings, Sixteenth International Conference on Automated Planning and Scheduling, 2006(January), 212–221. https://doi.org/10.1016/j.engappai.2023.106275

D.3 Area 3 – Category Theory

This reading list focuses on functorial data migration and graph rewriting.

1. Brown, K., Patterson, E., & Hanks, T. (2022). Computational Category-Theoretic Rewriting (Vol. 1). Springer International Publishing. https://doi.org/10.1007/978-3-031-09843-7

2. Patterson, E., Lynch, O., & Fairbanks, J. (2021). Categorical Data Structures for Technical Computing. Compositionality, 4(5), 1–27. http://arxiv.org/abs/2106.04703

3. Spivak, D. I. (2012). Functorial data migration. Information and Computation, 217, 31–51. https://doi.org/10.1016/j.ic

4. Patterson, E. (2017). Knowledge Representation in Bicategories of Relations. CoRR, 1–7 (Chp 5-6). http://arxiv.org/abs/1706.00526

5. Spivak, D. I., & Kent, R. E. (2012). Ologs: A categorical framework for knowledge representation. PLoS ONE, 7(1). https://doi.org/10.1371/journal.pone.0024274

Bibliography

[1] Samson Abramsky and Bob Coecke. Categorical Quantum Mechanics. *Handbook of Quantum Logic and Quantum Structures vol II*, pages 1–63, 2008.

[2] Christopher Agia, Krishna Murthy Jatavallabhula, Mohamed Khodeir, Ondrej Miksik, Vibhav Vineet, Mustafa Mukadam, Liam Paull, and Florian Shkurti. TASKOGRAPHY: Evaluating robot task planning over large 3D scene graphs. *5th Conference on Robot Learning*, (CoRL):1–18, 2021.

[3] Angeline Aguinaldo, Jacob Bunker, Blake Pollard, Ankit Shukla, Arquimedes Canedo, Gustavo Quiros, and William Regli. RoboCat: A Category Theoretic Framework for Robotic Interoperability Using Goal-Oriented Programming. *IEEE Transactions on Automation Science and Engineering*, pages 1–9, 2021.

[4] Angeline Aguinaldo and William Regli. A Graphical Model-Based Representation for PDDL Plans using Category Theory. In *ICAPS 2021 Workshop XAIP*, 2021.

[5] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. The 2011 International Planning Competition Description of Participating Planners Deterministic Track. *The 2011 International Planning Competition - Description of Participating Planners - Deterministic Track*, pages 58–60, 2011.

[6] Iro Armeni, Jerry Zhi-yang He, Junyoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3D Scene Graph. *Iccv*, 2019.

[7] Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009.

[8] J. Baez and M. Stay. Physics, topology, logic and computation: A Rosetta Stone. *Lecture Notes in Physics*, 813:95–172, 2011.

[9] Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1808–1815, 2007.

[10] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter Patel-Schneijder, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Recommendation, World Wide Web Consortium (W3C), February10 2004. See http://www.w3.org/TR/owl-ref/.

[11] Michael Beetz, Daniel Bessler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoglu, and Georg Bartels. Know Rob 2.0 - A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 512–519, 2018.

[12] Michael Beetz, B Ferenc, Nico Blodow, and Daniel Nyga. ROBOSHERLOCK : Unstructured Information Processing for Robot Perception. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1549–1556, 2015.

[13] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Ruhr, and Moritz Tenorth. Robotic roommates making pancakes. *IEEE-RAS International Conference on Humanoid Robots*, pages 529–536, 2011.

[14] Simon Bøgh, Oluf Skov Nielsen, Mikkel Rath Pedersen, Volker Krüger, and Ole Madsen. Does your Robot have Skills? *Proceedings of the 43rd International Symposium on Robotics*, 2012.

[15] Spencer Breiner, Al Jones, and Eswaran Subrahmanian. Categorical models for process planning. *Computers in Industry*, pages 1–35, 2016.

[16] Spencer Breiner, Blake Pollard, and Eswaran Subrahmanian. Functorial Model Management. *Proceedings of the Design Society: International Conference on Engineering Design*, 1(1):1963–1972, 2019.

[17] Spencer Breiner, Eswaran Subrahmanian, and Ram D. Sriram. Modeling the Internet of Things: A foundational approach. *ACM International Conference Proceeding Series*, Part F1271:38–41, 2016.

[18] Kristopher Brown, Evan Patterson, and Tyler Hanks. *Computational Category-Theoretic Rewriting*, volume 1. Springer International Publishing, 2022.

[19] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcís Palomeras, Natàlia Hurtós, and Marc Carreras. Rosplan: Planning in the robot operating system. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2015-January:333–341, 2015.

[20] Andrea Censi. A Mathematical Theory of Co-Design. Technical report, 2015.

[21] Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. A Comprehensive Survey of Scene Graphs: Generation and Application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):1–26, 2023.

[22] Yuxiao Chen, Ugo Rosolia, and Aaron D. Ames. Decentralized Task and Path Planning for Multi-Robot Systems. *IEEE Robotics and Automation Letters*, 6(3):4337–4344, 2021.

[23] Neil T. Dantam, Swarat Chaudhuri, and Lydia E. Kavraki. The Task-Motion Kit: An Open Source, General-Purpose Task and Motion-Planning Framework. *IEEE Robotics and Automation Magazine*, 25(3):61–70, 2018.

[24] Zinovy Diskin, Tom Maibaum, and Krzysztof Czarnecki. Intermodeling, queries, and kleisli categories. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7212 LNCS:163–177, 2012.

[25] Zoltan Dobra and Krishna S. Dhir. Technology jump in the industry: human–robot cooperation in production. *Industrial Robot*, 47(5):757–775, 2020.

[26] Stefan Edelkamp and Jörg Hoffmann. PDDL2 . 2 : The Language for the Classical Part of the 4th International Planning Competition 1 Introduction 2 Derived Predicates. *Syntax*, (195):1–21, 2004.

[27] Samuel Eilenberg and Saunders MacLane. General Theory of Natural Equivalences. *Transactions of the American Mathematical Society*, 58(2):231, 1945.

[28] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971.

[29] Maria Fox, Gerevini Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. *ICAPS 2006 - Proceedings, Sixteenth International Conference on Automated Planning and Scheduling*, 2006(January):212–221, 2006.

[30] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[31] Santiago Franco, Levi H S Lelis, Mike Barley, Stefan Edelkamp, Moises Martinez, and Ionut Moraru. The Complementary2 Planner in the IPC 2018. *International Planning Competition 2018 Classical Tracks*, pages 3–6, 2018.

[32] Cipriano Galindo. Planning Through World Abstraction. 20(4):677–690, 2004.

[33] Cipriano Galindo, Juan Antonio Fernández-Madrigal, Javier González, and Alessandro Saffiotti. Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966, 2008.

[34] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFRob: An efficient heuristic for task and motion planning. *Springer Tracts in Advanced Robotics*, 107:179–195, 2015.

[35] Alfonso Gerevini and Derek Long. Preferences and Soft Constraints in PDDL3. *Proceedings of the ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning*, 2006.

[36] Malik Ghallab, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - The Planning Domain Definition Language (Version 1.2). Technical report, Yale Center for Computational Vision and Control, 1998.

[37] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, 2004.

[38] Robert P Goldman and Ugur Kuter. Hierarchical Task Network Planning in Common Lisp: the case of SHOP3. *Proceedings of ELS '19: European Lisp Symposium (ELS '19)*, pages 73–80, 2019.

[39] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3-4):287–313, 1996.

[40] Micah Halter, Christine Herlihy, and James Fairbanks. A compositional framework for scientific model augmentation. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 323:172–182, 2020.

[41] Marc Hanheide, Moritz Göbelbecker, Graham S. Horn, Andrzej Pronobis, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Charles Gretton, Richard Dearden, Miroslav Janicek, Hendrik Zender, Geert Jan Kruijff, Nick Hawes, and Jeremy L. Wyatt. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*, 247:119–150, 2017.

[42] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. 20:291–341, 2003.

[43] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation. *Journal of Artificial Intelligence Research*, 14:263–312, 2001.

[44] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Computing Surveys*, 54(4), 2021.

[45] Adele E. Howe and Eric Dahlman. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, 17:1–33, 2002.

[46] Juraj Hromkovič and Nielsen Mogens. Graphs , Typed Graphs , and the Gluing. In *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, Berlin, Heidelberg, 2006.

[47] De An Huang, Danfei Xu, Yuke Zhu, Animesh Garg, Silvio Savarese, Li Fei-Fei, and Juan Carlos Niebles. Continuous Relaxation of Symbolic Planner for One-Shot Imitation Learning. *IEEE International Conference on Intelligent Robots and Systems*, pages 2635–2642, 2019.

[48] Ziyuan Jiao, Yida Niu, Zeyu Zhang, Song Chun Zhu, Yixin Zhu, and Hangxin Liu. Sequential Manipulation Planning on Scene Graph. *IEEE International Conference on Intelligent Robots and Systems*, 2022-October(i):8203–8210, 2022.

[49] Cliff A. Joslyn, Lauren Charles, Chris Deperno, Nicholas Gould, Kathleen Nowak, Brenda Praggastis, Emilie Purvine, Michael Robinson, Jennifer Strules, and Paul Whitney. A sheaf theoretical approach to uncertainty quantification of heterogeneous geolocation information. *Sensors (Switzerland)*, 20(12):1–36, 2020.

[50] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, pages 5678–5684, 2013.

[51] Craig Knoblock, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David E Smith, Ying Sun, and Daniel Weld. PDDL — The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control, 1998.

[52] Mieczyslaw M. Kokar, Kenneth Baclawski, and Hongge Gao. Category theory-based synthesis of a higher-level fusion algorithm: An example. *2006 9th International Conference on Information Fusion, FUSION*, 2006.

[53] S. P. Kovalyov. Category-Theoretic Approach to Software Systems Design. *Journal of Mathematical Sciences (United States)*, 214(6):814–853, 2016.

[54] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

[55] Tom Leinster. *Basic Category Theory*. Cambridge University Press, 2016.

[56] Haizhen Li and Xilun Ding. Adaptive and intelligent robot task planning for home service: A review. *Engineering Applications of Artificial Intelligence*, 117(November 2022):105618, 2023.

[57] Gi Hyun Lim, Il Hong Suh, and Hyowon Suh. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, 41(3):492–509, 2011.

[58] Weiyu Liu, Angel Daruna, Maithili Patel, Kartik Ramachandruni, and Sonia Chernova. A survey of Semantic Reasoning frameworks for robotic systems. *Robotics and Autonomous Systems*, 159:104294, 2023.

[59] Tomás Lozano-PÉrez. Robot Programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

[60] Peter Lefanu Lumsdaine and Michael Shulman. Semantics of higher inductive types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 169(1):159–208, 2020.

[61] Saunders MacLane. *Categories for the Working Mathematician*. Springer New York, NY, 1971.

[62] Mau Magnaguagno. HyperTensioN, 2021.

[63] Jade Master, Evan Patterson, Shahin Yousfi, and Arquimedes Canedo. String Diagrams for Assembly Planning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2020.

[64] Runqing Miao, Qingxuan Jia, and Fuchun Sun. Long-term robot manipulation task planning with scene graph and semantic knowledge. *Robotic Intelligence and Automation*, (November), 2023.

[65] Dejan Pangercic, Moritz Tenorth, Dominik Jain, and Michael Beetz. Combining perception and knowledge processing for everyday manipulation. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, (Section III):1065–1071, 2010.

[66] A. K. Pani and G. P. Bhattacharjee. Temporal representation and reasoning in artificial intelligence: A review. *Mathematical and Computer Modelling*, 34(1-2):55–80, 2001.

[67] Evan Patterson. Knowledge Representation in Bicategories of Relations. *CoRR*, pages 1–73, 2017.

[68] Evan Patterson, Owen Lynch, and James Fairbanks. Categorical Data Structures for Technical Computing. *Compositionality*, 4(5):1–27, 2021.

[69] David Paulius and Yu Sun. A Survey of Knowledge Representation in Service Robotics. *Robotics and Autonomous Systems*, 118:13–30, 2019.

[70] Michael Robinson. Sheaves are the canonical data structure for sensor integration. *Information Fusion*, 36:208–224, 2017.

[71] Stuart Russell, Peter Norvig, Ming-Wei Chang, Jacob Devlin, Anca Dragan, David Forsyth, Ian Goodfellow, Jitendra Malik, and Vikash Mansinghka. *Artificial Intelligence: a modern approach*. Pearson, 4th editio edition, 2021.

[72] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks. *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 13B:11962–11971, 2021.

[73] Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1):4444–4451, 2017.

[74] David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012.

[75] David I Spivak and Robert E Kent. Ologs: a categorical framework for knowledge representation. *PLOS ONE*, 2011.

[76] David I. Spivak and Ryan Wisnesky. Relational foundations for functorial data migration. *DBPL 2015 - Proceedings of the 15th Symposium on Database Programming Languages*, pages 21–28, 2015.

[77] Moritz Tenorth and Michael Beetz. Representations for robot knowledge in the KNOWROB framework. *Artificial Intelligence*, 247:151–169, 2017.

[78] Moritz Tenorth, Alexander Clifford Perzylo, Reinhard Lafrenz, and Michael Beetz. The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments. *Proceedings - IEEE International Conference on Robotics and Automation*, (3):1284–1289, 2012.

[79] Moritz Tenorth, Lars Kunze, Dominik Jain, and Michael Beetz. KNOWROB-MAP – Knowledge-Linked Semantic Object Maps. *IEEE-RAS International Conference on Humanoid Robots*, pages 430–435, 2010.

[80] Sylvie Thiebaux, Jorg Hoffmann, and Bernhard Nebel. In Defense of PDDL Axioms. *Syntax And Semantics*, pages 1–6, 2000.

[81] Simon Thompson. The Craft of Functional Programming (2nd Edition). *Addison Wesley*, page 512, 1999.

[82] Karthik Mahesh Varadarajan and Markus Vincze. AfRob: The affordance network ontology for robots. *IEEE International Conference on Intelligent Robots and Systems*, pages 1343–1350, 2012.

[83] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55(March):248–266, 2018.

[84] W3C. Rdf 1.1 concepts and abstract syntax. https://www.w3.org/TR/rdf11-concepts/, 2014. Accessed: yyyy-mm-dd.

[85] Haoqi Wang, Vincent Thomson, and Chengtong Tang. Change propagation analysis for system modeling using Semantic Web technology. *Advanced Engineering Informatics*, 35(May 2017):17–29, 2018.

[86] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[87] Shiqi Zhang, Michael Gelfond, and Jeremy Wyatt. An Architecture for Knowledge Representation and Reasoning in Robotics. *Lecture Notes in Computer Science*, (Ll):1–12, 2012.