

Dialogue-driven Multi-Agent Activity Planning

by

Anoopkumar S. Sonar

B.S.E. Princeton University (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

© 2024 Anoopkumar S. Sonar. This work is licensed under a [CC BY-NC-ND 4.0](#) license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Anoopkumar S. Sonar
Department of Electrical Engineering and Computer Science
May 17, 2024

Certified by: Brian C. Williams
Professor of Aeronautics and Astronautics, Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Dialogue-driven Multi-Agent Activity Planning

by

Anoopkumar S. Sonar

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

ABSTRACT

A fundamental challenge in robotics is to build a general-purpose system with multiple agents that can perform a wide range of tasks based on specifications provided in natural language. This work presents a novel dialogue-driven activity planning framework for multi-agent scenarios. We present a method that accepts commands from a user in natural language and translates it to an intermediate form called a state plan by leveraging large language models. We further experiment with chain-of-thought prompting to improve the translation from natural language to state plans. In conjunction with an action model, this state plan is utilized by a constraint-based generative planner called ctBurton which outputs a full grounded plan in the form of a state and control trajectory. We demonstrate the utility of our method across three different scenarios – a presentation system, search-and-rescue, and multi-agent assembly – along with experiments on its scalability.

Thesis supervisor: Brian C. Williams

Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to begin by thanking my advisor Professor Brian Williams for providing me the opportunity to work with the MERS group, and the rest of the lab for helping me get up to speed. I would also like to thank David Wang and Andreas Hofmann for their advice and mentorship throughout the process.

I would like to thank all my labmates – Viraj, Marlyse, Yuening, Jake, Shashank, Amy and Aneesh for warmly welcoming me into the group and for the many fruitful and productive conversations over the past two years. I will cherish all the brainstorming sessions I had with Viraj in the colony for years to come. Marlyse and Yuening played a key role in helping me get up to speed on all of the lab’s past work without which it would’ve been difficult to finish my research work.

More broadly, I’d like to thank my many friends across MIT (Artem, Ishaan, Akarsh, Abhi, Ryan), in Boston (Nathaniel, Jasper, Henry), and beyond (Amey, Rohan, Himanshu, Tanay) for keeping me sane and making the graduate student experience fun and memorable.

Finally, I would like to thank my parents for the love and support that they’ve always given me, and for believing in me through the years without which I never would have made it this far.

Contents

Title page	1
Abstract	3
Acknowledgments	5
List of Figures	11
List of Tables	13
1 Introduction	15
1.1 Overview	16
1.2 Motivating examples	16
1.2.1 Presentation System : Computer-Connection-Projector	17
1.2.2 Multi-Agent Assembly	18
1.2.3 Search-and-Rescue operations	20
1.3 Statement of Contributions	21
1.4 Thesis Outline	23
2 Related Work	25
2.1 Natural language to robot plans	25
2.2 Generative Temporal planners	27
3 CTBurton Planner	29
3.1 Overview	29
3.1.1 Action Model	29
3.1.2 State Plan	30
3.2 Definitions	30
3.2.1 PCTA State Planning Problem	31
3.2.2 Action Model	31
3.2.3 State Plan	34
3.3 Encoding - Decision Variables	36
3.4 Encoding - Constraints	40
3.4.1 Event constraints	40
3.4.2 Clock or clock parameters	40
3.4.3 Ordering of events	40

3.4.4	Clock variable and clock offset consistency	40
3.4.5	Transitions within a stage	40
3.4.6	Transitions between stages and associated resets	41
3.4.7	Making clock variables and parameters persist	42
3.4.8	Constraints that ensure determinate transitions	43
3.4.9	Constraints that ensure concise runs	43
3.5	Encoding - State Plan	45
3.5.1	Encoding Initial State	45
3.5.2	End State Accepting Automaton	45
3.5.3	Episode Accepting Automaton	46
3.5.4	Temporal Constraint Accepting Automaton	46
3.5.5	Event Accepting Automaton	47
3.6	Encoding - Objective/Cost Function	47
3.6.1	Optimization Problem Formulation	48
3.6.2	Temporal Optimal Satisfiability (TOP) problem	48
3.6.3	Deterministic Mixed Logic Temporal Problem	48
3.7	Implementation Details	49
4	Natural Language to State Plans	51
4.1	JSON specifications	51
4.1.1	System config	52
4.1.2	Experiment Config	52
4.1.3	State Plan	54
4.2	Prompting approaches	55
4.2.1	Direct JSON generation	55
4.2.2	Chain-of-Thought (CoT) prompting	55
5	Experimental Results	59
5.1	Computer-Connection-Projector	59
5.1.1	Initial in-context learning prompt	59
5.1.2	Direct JSON generation	61
5.1.3	Chain-of-thought prompting	62
5.1.4	Test prompts	62
5.2	Multi-Agent Assembly	67
5.2.1	Initial in-context learning prompts	67
5.2.2	Test prompt	68
5.3	Search-and-Rescue	69
5.3.1	Initial in-context learning prompts	69
5.3.2	Test prompts	70
5.4	Performance assessment	71
6	Conclusion and Future Work	73
6.1	Takeaways	73
6.2	Limitations	74
6.3	Future Work	75

7	Appendix	77
7.1	Code	77
7.2	Output decision variables	77
7.2.1	Presentation System	77
7.2.2	Multi-agent Assembly	83
7.2.3	Search-and-Rescue	89
	References	97

List of Figures

1.1	The action model for the presentation system example consisting of the computer, connection and projector.	17
1.2	The state plan for the presentation system corresponding to the text input specified above.	18
1.3	The full state and control trajectory that is output by CTBurton for the action model and the state plan presented above.	19
1.4	An illustration of the multi-agent assembly scenario. The goal of the two agents is to re-arrange the blocks from their initial states to the final states.	20
1.5	The action model for the two robot arms in the multi-agent assembly example.	21
1.6	Demonstration of the multi-agent assembly scenario on physical hardware from prior research work [3]	22
1.7	An illustration of the search-and-rescue scenario.	23
1.8	The action model for the ground agents in the search-and-rescue example.	24
3.1	The connection automaton in the presentation example.	30
4.1	Chain of thought prompting relies on breaking down a larger problem into smaller subproblems.	56

List of Tables

5.1	Computation Time (in seconds) for various problem scales for the presentation example	71
-----	---	----

Chapter 1

Introduction

A longstanding goal in robotics (and a dream for roboticists) is to build a general purpose robotic system that can perform a wide range of tasks in their immediate environments (offices and homes, to name a few) to improve our quality of life, enhance productivity and make our lives more convenient.

Broadly speaking, roboticists have broken up this problem into two key components.

1. **Higher level** - Activity or Task planning

This deals with the sequencing and scheduling of various activities that an agent can perform in order to achieve some larger objective. For instance, you might want your robot to bring you a can of coke from the fridge.

Activity planning tackles the problem of generating a sequence of steps such as 1) going to the fridge, 2) opening the door, 3) picking up the can, 4) bringing it back to the person.

To put it simply the objective of activity planning is, given an end goal and a set of high level primitives/actions (eg., goto(position), pick(object)) that we know the agent is capable of, to generate this sequence of steps that transforms the initial state (where the person *doesn't* have a can of coke) into the goal state (where the person *does* have a can of coke).

This sequence of steps must also often satisfy a set of constraints on the order and timing at which specific events occur and the activity/task planning method must take these into account.

2. **Lower level** - Motion planning

This deals with the specific kinematics and dynamics of the agents involved in order to successfully complete a particular task, such as picking up an object while respecting certain constraints (such as collision avoidance). These details are often abstracted away from the activity planner. Depending on the specific agent, accurate and fast

motion planning can be fairly non-trivial, such as in the case of mobile manipulators, which can have more than 10 degrees of freedom.

The motion planner must find the time trajectory of the various joint angles or actuator inputs in order to generate a motion that completes the appropriate higher level action (such as pick, goto).

Depending on the exact method used, this trajectory can be discretized in time (as a list with each element corresponding to the agent’s pose at a particular time), or be represented in a continuous form using polynomials/bezier-splines. This is then passed on to a trajectory tracking control method, which ensures that the motion plan is executed successfully.

There also exist several families of methods that solve the two problems jointly, including hybrid discrete / continuous planning[1], task and motion planning (TAMP)[2] and combined activity or motion planning (CAMP)[3].

1.1 Overview

The goal of this thesis is to leverage recent advances in natural language processing, specifically the advent of large language (‘foundation’) models, to build a natural language friendly **activity planning framework for multi-agent scenarios**.

This framework **accepts commands** from the user in **natural language** ("*get me a can of coke*") and translates it to a scaffolding or plan sketch, called a **state plan**, which we define precisely in a later section, that is composed of primitives from a language of qualitative constraints. This language includes qualitative and quantitative temporal relationships across events in the given scenario.

A state plan is utilized in conjunction with an **action model**, which represents the set of possible actions and states that a group of agents can be in, by a planner called **ctBurton** (a constraint-based generative planner) to generate a full state and control trajectory as it varies with time. This trajectory can consequently be consumed by a variety of existing motion planners.

Finally, we demonstrate the performance of this method on a few different motivating examples that are introduced in the next section.

1.2 Motivating examples

In this section, we introduce a few different motivating examples to demonstrate how our framework can be utilized in practice.

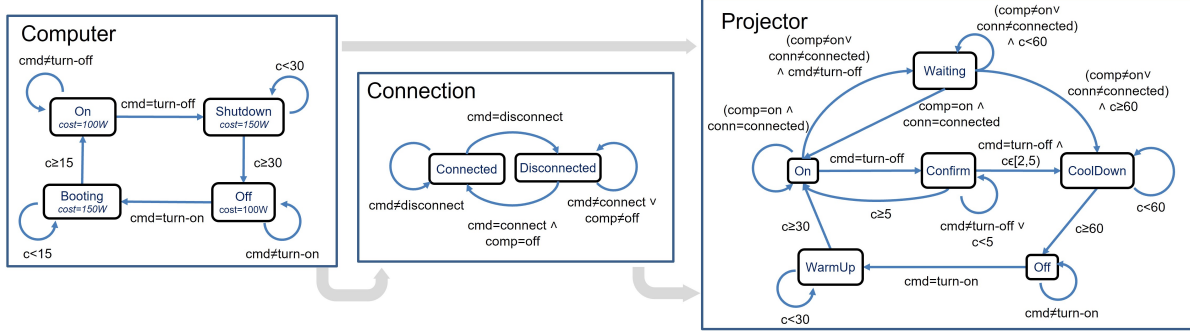


Figure 1.1: The action model for the presentation system example consisting of the computer, connection and projector.

1.2.1 Presentation System : Computer-Connection-Projector

Let's consider a scenario where our goal is to control a presentation system. This system consists of a computer, a projector and a connection between them (taken from [4]). The computer needs some time to boot up from it's off state and some time to shutdown once it's on. In a similar manner, the projector takes time to warm up when it's turned on, and to cool down once it's turned off. We also know that the projector cannot be turned on until a connection is established, and the projector shuts down automatically when disconnected from the computer.

All these discrete states, such as (*on*, *booting*, *off*), that the agents can be in are called *locations*. The *transitions* between these states and the conditions that need to be satisfied for those transitions to occur are encoded in the **action model** of the system.

Note In this work, we assume that an action model is already provided to us, i.e. it is not learned or obtained from a large language model. However, learning this model or obtaining it from a pre-trained model is an exciting direction for future research.

This **action model** is formalized using a *parameterized concurrent timed automata (PCTA)*[5]. We call it *parameterized* because the various guard conditions that allow or disallow certain transitions can be functions of other location variables or random variables drawn from some statistical distribution. It's called *concurrent* because the *PCTA* consists of multiple different state machines that represent multiple agents in the environment that can perform actions and interact with each other simultaneously. We define these terms more rigorously in a future chapter.

A visualization of the action model for this presentation system example is shown in Figure 1.1.

State Plan The user might provide a text input of the following form

The computer and projector start in the off state, and the connection is disconnected. The projector should turn on within 60 seconds and stay on for exactly

1800 seconds, following which everything should turn off within 600 seconds.

We aim to leverage large language models such as ChatGPT by OpenAI to translate this text input to an intermediate state plan which can be parsed by our CTBurton planner. The planner's output is the full state and control trajectory which is guaranteed to satisfy the action model and the constraints specified by the state plan.

A visualization of the state plan for the user input in natural language mentioned above is shown in Figure 1.2

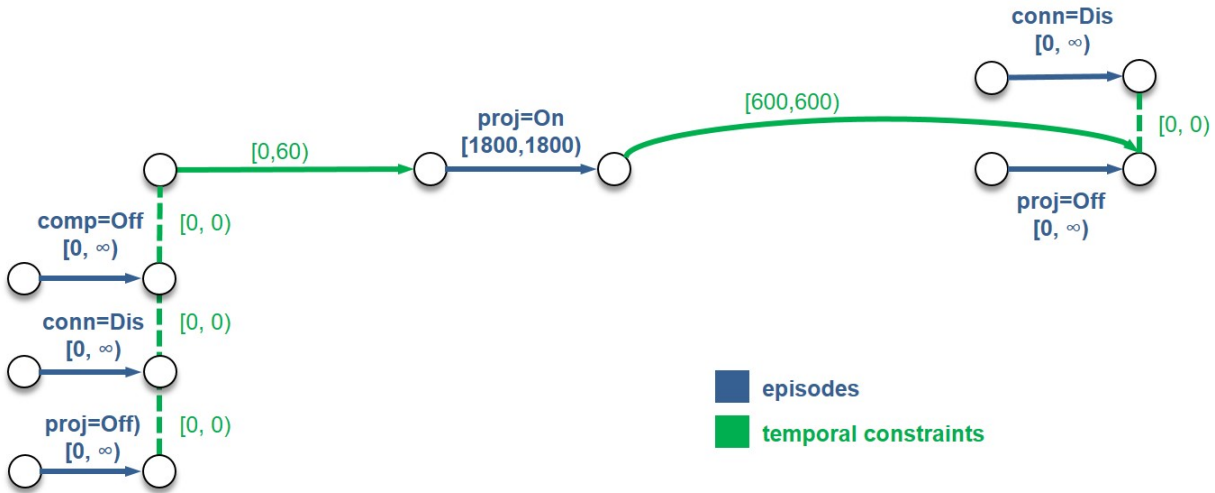


Figure 1.2: The state plan for the presentation system corresponding to the text input specified above.

Output Given the action model and the state plan, the goal of our planner CTBurton is to generate a *grounded plan*, i.e. the **full state and control trajectory** which fully describes the location/state of each automaton at every time step and the actions that are performed at discrete time points.

This final output is visualized in Figure 1.3

1.2.2 Multi-Agent Assembly

The next example involves using two 6/7-dof robotic manipulators that assemble a set of parts into a final end-product as specified by a CAD model as done before in [3].

Imagine that you want to solve a block stacking task as illustrated in Figure 1.4. The initial and the final states of the objects are specified as a kinematic graph describing the relationship between those objects using keywords such as *above* and *below*. Initially, both the robots are in the *home* location. The two agents must plan when to pick up and place certain objects and in what order in order to solve the task. Each arm only has one gripper and can only hold one block at a time. This action model is visualized in Figure 1.5.

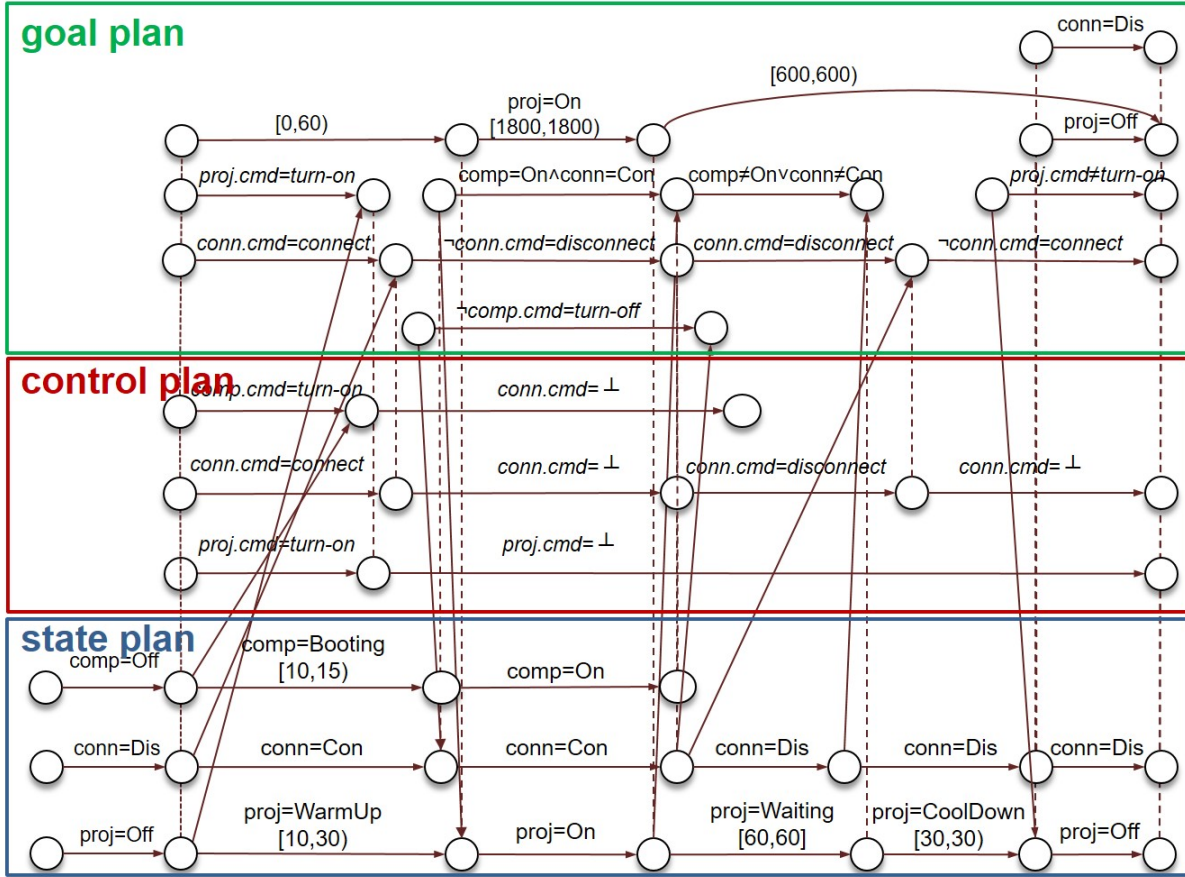


Figure 1.3: The full state and control trajectory that is output by CTBurton for the action model and the state plan presented above.

The two actions that each robot can perform are *pick* and *place* whose pre-conditions and post-conditions can be written in a parametric form in terms of the object that is being picked or placed as well as the object it's picked up from or placed on. For instance, we cannot pick a lego block that has another lego block on top, and cannot place a lego block A on top of lego block B, if B has another block on top of it already. The action model accounts for these limitations using guard constraints on transitions.

State Plan Natural language input of the following form is provided by the user -

Both robot arms start in the home state. Block C is on the floor. Block B is stacked on top of block C, and block A is stacked on top of block B. Pick up block A and place it on the floor.

The state plan is consequently consumed by CTBurton in order to produce the full state and control trajectory.

In order to be deployed in the real world, this state and control trajectory could be integrated with other work on *roadmap generation* algorithms whose output is consequently



Figure 1.4: An illustration of the multi-agent assembly scenario. The goal of the two agents is to re-arrange the blocks from their initial states to the final states.

consumed by a *multi-agent path finding* (MAPF) algorithm which performs the lower-level motion planning [3]. The motion planner can also be used to verify the feasibility of the higher level plan and if CTBurton’s plan causes collision or isn’t feasible dynamically, this conflict can be returned to CTBurton so that it can generate a new plan accounting for those constraints.

1.2.3 Search-and-Rescue operations

Consider a scenario with a heterogeneous set of robots consisting of drones and ground/terrestrial vehicles that are attempting to rescue a group of humans scattered across rooms in a building that has caught fire. A simple illustration of this is provided in Figure 1.7.

We assume that the swarm of drones has already mapped the building and identified the location of the various humans. The ground vehicles must now enter the building, navigate to the various humans, pick them up in order to rescue them. A ground vehicle cannot enter a room until one of the doors has been opened.

Each ground agent is capable of rescuing one person. As the agents navigate the environment, they must open doors in order to enter rooms, pick up the humans. The various ground agents may start off in different initial positions. For the sake of simplicity, we assume that all the ground agents have the same action model. In our scenario, we consider two ground agents. Their action model is visualized in Figure 1.8.

State Plan Based on the illustrated scenario, the natural language input of the following form could be provided by the user -

Person 1 and Person 2 are in room 2 and must both be rescued. Person 1 must be rescued before person 2.

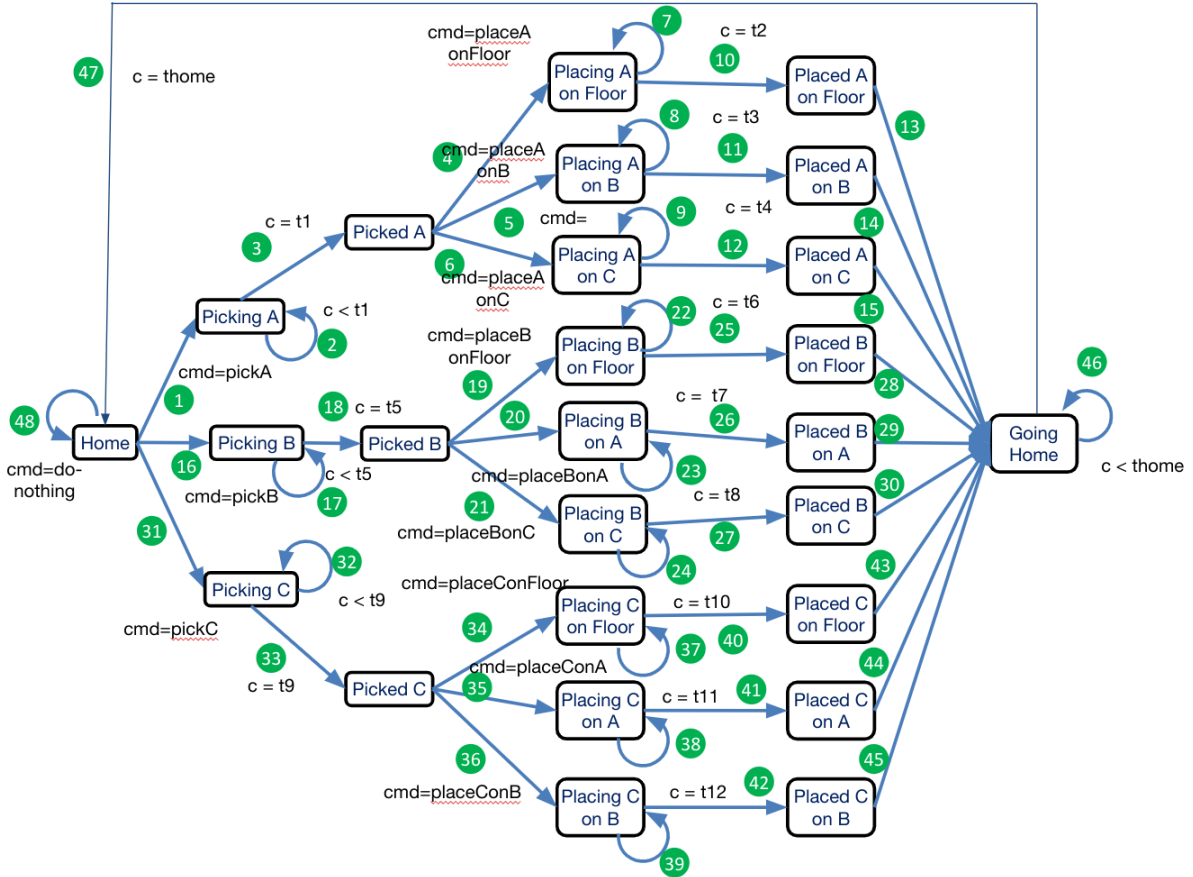


Figure 1.5: The action model for the two robot arms in the multi-agent assembly example.

This natural language input is mapped onto a state plan. The state plan is consequently consumed by CTBurton in order to produce the full state and control trajectory.

In order to be deployed in the real world, this state and control trajectory could be integrated with a standard motion planner for point-to-point navigation or rely on end-to-end learned motion primitives.

1.3 Statement of Contributions

In this thesis, we present a novel dialogue-driven multi-agent activity-planning framework. The central problem tackled in this work is that of accepting user commands in natural language and turning them into precise state and control trajectories for multiple agents as specified by the user. We approach this problem by taking a two-step approach that integrates two different modules, 1) a LLM-based translator that maps from natural language to an intermediate representation called a state plan, and 2) a generative planner called CTBurton that turns a state plan into a grounded plan i.e. the full state and control trajectory. Relative to prior works that directly output action plans (that may or may not be grounded) or use task specifications like temporal logic that are not friendly to the layperson, we use a state plan which is a more intuitive way of specifying tasks. Translating to a

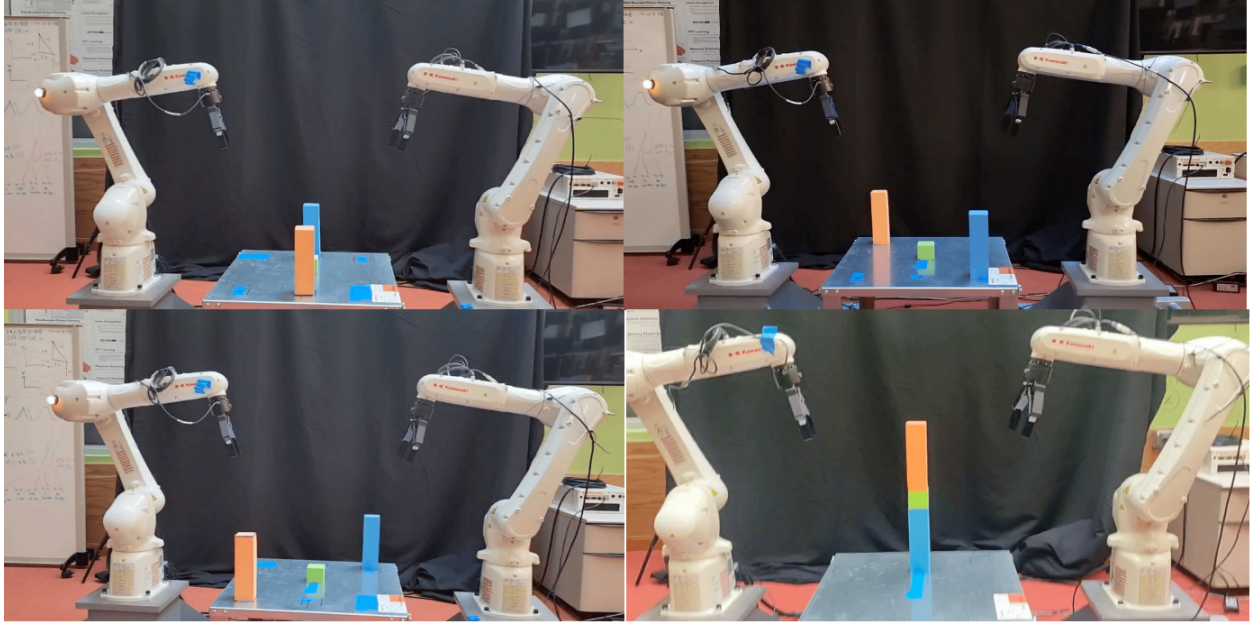


Figure 1.6: Demonstration of the multi-agent assembly scenario on physical hardware from prior research work [3]

state plan and performing generative planning as opposed to directly producing plans using LLMs guarantees completeness and soundness of plans. As a result, our method provides a dialogue-friendly way for humans to safely interact with multi-agent robotic systems.

The key contributions of this thesis are to -

1. **Propose** a novel neuro-symbolic approach for obtaining full state and control plans from natural language user commands with state plans as an intermediate task specification.
2. **Implement** the deterministic¹ minizinc encoding for the CTBurton planner in order to formulate the planning problem as a mixed logic linear program (MLLP).
3. **Generalize** this minizinc encoding and write a python wrapper around it such that CTBurton can be used with any arbitrary specification of a state plan and an action model.
4. **Design** and implement a JSON specification format for the state plan which is compiled into a minizinc encoding.
5. **Experiment** with multiple prompting methods for large language models such as ChatGPT in order to translate natural language input to the state plan JSON specification.

¹This encoding is deterministic because it doesn't account for uncertainty in the clock parameters that are introduced in CTBurton.

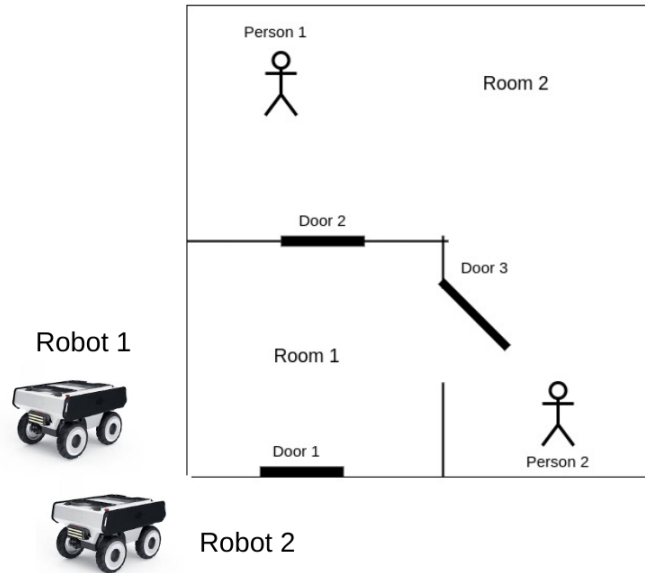


Figure 1.7: An illustration of the search-and-rescue scenario.

6. **Demonstrate** the output and empirical performance of CTBurton on the different motivating examples defined in the previous section.

1.4 Thesis Outline

This chapter introduced the concepts of activity and motion planning, the goal of this thesis and a few motivating examples, and finally summarized the key contributions.

In Chapter 2, we look at the related past work that has been done in using natural language in robot planning, as well as in generative temporal planning. We also highlight some of the advantages of using our approach over the other methods.

In Chapter 3, we introduce the CTBurton planner, formalize and define the various concepts involved, and introduce its optimization encoding. The encoding is specified by defining the various decision variables that are necessary and the constraints over them.

In Chapter 4, we dive deeper into the exact method for prompting large language models (LLMs) in order to generate the state plan JSONs from natural language user commands.

In Chapter 5, we present the experimental results when our method is applied to solve the various motivating examples mentioned in this chapter. We demonstrate the generalization ability that is unlocked by using our method and point out some of its limitations/drawbacks.

Finally, Chapter 6 provides concluding thoughts on the work done in this thesis and potential research directions for future work.

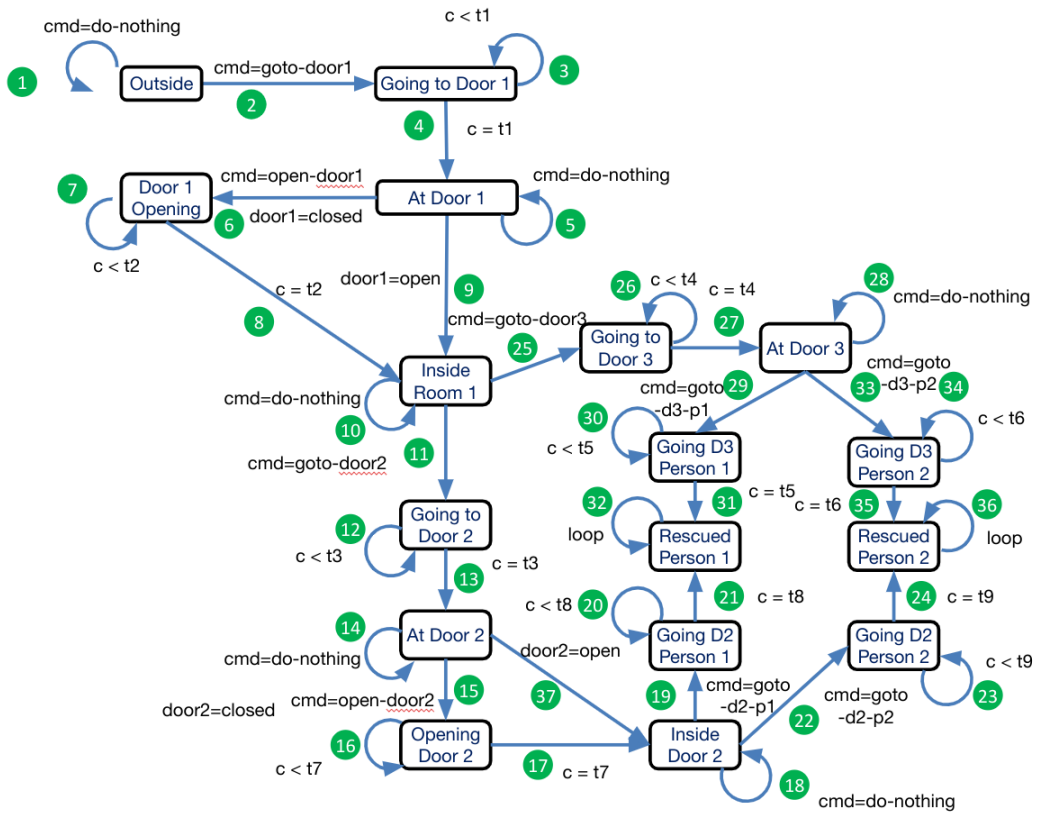


Figure 1.8: The action model for the ground agents in the search-and-rescue example.

Chapter 2

Related Work

The novel approach we present in this thesis integrates two key modules – the LLM-based text to state plan translator, and the state plan to grounded plan generative planner. In this chapter we briefly review some of the related and prior work which inspired the directions in this thesis, and the relative advantages and disadvantages of our method.

The approach we present in this thesis primarily builds upon Uhura [6], [7], an interactive decision support system to help human-robot teams resolve over-subscribed planning problems. The core of Uhura is a conflict-directed diagnosis algorithm that can detect conflicts between planning goals and incomplete planning domains. To address the conflicts and incompleteness, the algorithm needs to generate alternative plans by relaxing some of the goals and domains. Once a feasible alternative is found, Uhura will initialize a dialog with the user to explain the cause of failure and present the alternative. The user can either take Uhura’s advice, or ask Uhura to find different alternatives.

Uhura accepts natural language input and returns a hybrid activity/motion plan. It uses a TPN activity planner called Kirk which generalizes the job shop scheduling problem. As opposed to our approach, Uhura does not perform generative activity planning. The state plan used in Uhura is over continuous states and generates trajectories using a planner (pSulu) which is a predecessor to Magellan.

Our method does not consider relaxations or incompleteness of domains. However, we extend the action model formalism beyond temporal planning networks (TPNs) to be significantly more expressive using parameterized concurrent timed automata (PCTA). Moreover, we introduce a notion of a state plan to constrain the discrete state evolution of the multi-agent system based on the dialog input.

2.1 Natural language to robot plans

With the rise of the large language models, also sometimes referred to as “foundation” models, many interesting directions have emerged for translating natural language input to programs that can be executed on a robotic system.

One such example is ProgPrompt[8] which presents a programmatic LLM prompt structure that grounds its output by structuring the prompt to specify actions that are feasible given the environment. This work does not explicitly generate a state plan, but directly maps the input text to a sequence of action primitives that the agents are capable of. Since it also does not rely on a formal action model of the agents in the environment, the final output is not guaranteed to be correct.

In contrast, CTBurton will guarantee that the full state and control trajectory will respect the constraints of the action model and the state plan. The ProgPrompt plans are also not guaranteed to be optimal as opposed to CTBurton which outputs optimal plans (under certain restrictions). CTBurton is guaranteed to be *sound, complete and optimal* with respect to its problem statement i.e. the action model and the state plan provided.

Problem statements are an abstraction of the real world, and they can be misaligned. In practice, the state plan may not correctly interpret user intent, and the action model may be incomplete (not sufficiently expressive) or incorrect.

SayCan[9] launched a flurry of work that use large language models for long-time horizon planning tasks. SayCan uses a combination of the language model output probability and the value function of some pretrained skills to ground the plan. Specifically, the optimal next step to execute is chosen by ranking the product of the probability of the next step according to the large language model and the value function for each pre-trained skill. The most likely combination is then executed. Utilizing the value function is a way to “ground” the output action in the physical world since the method will never pick a skill to execute that has a very small value function given the specific context/environment.

Code-as-Policies[10] represents another approach that takes in some natural language input and directly outputs python code based on the action primitives mentioned in the context. Code-as-Policies also learns to define its own compositional primitives that it can then reuse when generating plans which can lead to very compact output plans. Both SayCan and Code-as-Policies suffer from the same issues as ProgPrompt such as the lack of soundness or optimality.

SayPlan[11] uses LLMs in combination with 3D scene graphs(3DSGs) and exploits the hierarchical nature of the 3DSGs. These 3D semantic graphs are hierarchical because they separate a physical environment into floors, rooms, and sections within rooms. The 3DSGs are used to perform a semantic search for task-relevant subgraphs to enable better planning over longer time horizons. The key contribution of SayPlan is to integrate a scene graph into the planning process which is complementary to our method and future work could easily incorporate a scene graph into the planning process as opposed to using our current representation of the initial and final state. The initial and the final state can be represented using a scene graph that accounts for relationships between entities in the environment as opposed to a flat list of variables. Since SayPlan also does not rely on a formal action model, it does not guarantee optimality.

ChatGPT for Robotics[12] performs additional exploration and outlines a strategy that combines prompt engineering and the creation of a high level library of primitives/skills that allow ChatGPT to adapt to various robotic tasks. This work’s key contribution is testing a wide variety of prompting methods and establishing good prompting practices which we utilize in our work. They also test performance with various plan representations such as free-form dialog, XML tags and python code.

NL2TL [13] provides a framework for transforming natural language commands to precise specifications using temporal logic. Finally, another fascinating line of work is AutoTAMP [14] which uses LLMs to perform a few-shot translation from natural language task descriptions to signal temporal logic as the intermediate task description that is consumed by a task and motion planning (TAMP) algorithm to jointly solve the task and motion plan. Our work does not solve the motion planning problem and focuses exclusively on the activity/-task planning problem.

For a more comprehensive summary of the role of foundation models in robotics we direct the readers to [15].

In this work, we introduce the notion of a **state plan** as the intermediate representation that the natural language input is converted to which in turn is utilized by a generative planner. We claim that laypersons will find state plans to be a more intuitive way of specifying tasks for a given system as compared to other methods like temporal logic which are primarily meant to be used for specifying safety or liveness properties. State plans require the human to express in natural language how the state of the system will evolve which is a more natural way of thinking about tasks.

2.2 Generative Temporal planners

This thesis presents an implementation and multiple demonstrations of the ctBurton planner which has the ability to model timed transitions, periodic transitions, concurrency of the various automata, time-evolved goals as well as controllable action durations. We claim that none of the existing planners support all of these capabilities simultaneously.

In standard timed automata, the transition time is typically a fixed constant and so is the amount of time that an automata spends in a particular location. CTBurton uses a clock parameter instead which can be used to dynamically change the transition time.

As opposed to some other classical planners, CTBurton also uses a state plan as opposed to a goal state which is significantly more expressive. As mentioned in the previous section, temporal logic is another tool that is used in the planning and logic-based control community for describing goals over time but it can be difficult to use for laypersons and is more expressive than is often needed to specify plans over time so CTBurton uses state plans instead which are similar to least-commitment plans that constrain state evolution as opposed to

actions. State plans serve as a scaffolding or a "sketch" of the behavior that the user wants when they haven't fleshed out the full details. State plans have been used extensively by continuous, discrete, and hybrid state planners such as Sulu[16], Chekov[17], [18], tBurton[4], Kongming[19], and cKongming[20].

The ctBurton planner solves the problem of combined activity planning and scheduling as a coupled problem. ctBurton has connections to three key bodies of prior work – PDDL durative planning [21]–[23], timeline-based planning[24]–[26], and finally controller synthesis based on timed automata [27]–[29].

A timeline is a complete history of the changes to the value of a variable (for instance the state of an automaton), as computable by a partially specified plan, over the specified planning horizon. In this sense, a timeline could be considered similar to a grounded plan given a partial state plan. Timed automata synthesis and several PDDL durative planners address activity planning but not scheduling. There also exist PDDL planners that offer flexibility in duration. The key difference between ctBurton and these alternatives mainly revolve around the specific search methods used and the expressiveness of the action model.

Timeline-based planners such as Europa[30] and Aspen[31] express a rich set of constraints on metric time and resources, and support combined planning and scheduling, but depend on domain-specific heuristics in order to efficiently guide backtrack or repair-based search.

While Burton [32], [33] does not need domain specific heuristics, it lacks the ability to reason over metric specifications of time. Metric temporal PDDL planners [34]–[36] based on heuristic forward search are domain-independent, fast and scalable, but often do not allow for flexibility in the duration of activities, do not account for uncertainty in actions and lack support for several important features listed above, such as timed and periodic transitions.

Chapter 3

CTBurton Planner

The goal of this chapter is to supplement the ctBurton [5] journal publication by summarizing the encoding in a form that makes it easy to implement without necessarily understanding all the theoretical details.

We accomplish this by - 1) describing the inputs and outputs of ctBurton intuitively, 2) describing the encoding in terms of these inputs, and 3) finally describe how to extract a solution from the solver output.

3.1 Overview

Recall from the introduction that ctBurton accepts two inputs - an *action model* of the system in the form of a parameterized concurrent timed automata (PCTA), and a *state plan* to output a *grounded state and control* trajectory.

3.1.1 Action Model

Intuitively, an automaton is specified by describing the discrete locations/states that it can be in, and the transitions between those states. The transitions might be subject to certain *guard* constraints or conditions in order for that transition to be active.

Here we present the connection automaton from the presentation example to demonstrate how to encode a single automaton in PCTA. The process for encoding the other automata is similar. The connection automaton is visualized in Figure 3.1.

As we can see, the connection automaton can be in two possible locations, *connected* or *disconnected*. There are 4 possible transitions in this automaton. We have two idle transitions and two state changing transitions.

In the idle transitions, the connection idles in it's current location. If the connection is connected, then it stays in that state as long as the control command is *not disconnect*. If the connection is disconnected, it stays disconnected as long as the control command is *not*

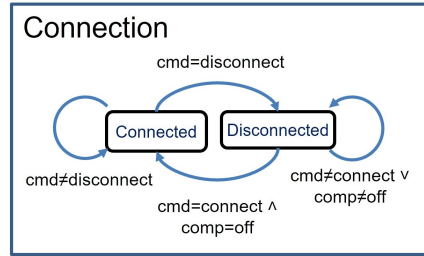


Figure 3.1: The connection automaton in the presentation example.

connect or the computer is *not* **off**. Note that the guard conditions for a transition in an automaton can be a function of the location of other automata (the computer in this case) in a PCTA.

In the state changing transitions, the connection transitions from the disconnected state to the connected state if the control command is **connect** and the computer is **off**. If the connection is in the connected state, it transitions to the disconnected state if the control command **disconnect** is executed.

3.1.2 State Plan

In ctBurton, the state plan is represented via a set of accepting automata. Specifically, we have an end state accepting automaton, episode accepting automata, temporal constraint accepting automata, and finally the event accepting automata.

In the state plan for the presentation example in Figure 1.2 we can see two different types of constraints on the state evolution. The episodes are presented in blue, and the temporal constraints in green. The first temporal constraint enforces that the projector should turn on within 60 seconds from when the clock starts. The second temporal constraint enforces that projector should turn off in exactly 600 seconds. Finally, the episode constraint between the two temporal constraints enforces that the projector should stay on for exactly 1800 seconds.

Each of these episodes and temporal constraints has a corresponding accepting automaton which has to move into the accepted state for the state plan to be satisfied by the grounded plan.

3.2 Definitions

We begin by introducing the core notation, terms and their definitions from ctBurton that are most relevant to the implementation and are used extensively in the rest of this thesis. This is **heavily adapted** from the future journal submission of the planner [5] to which we direct the reader for a more thorough treatment.

3.2.1 PCTA State Planning Problem

The central goal of ctBurton is to output a grounded plan given a state plan and an action model. We formalize this by saying that ctBurton’s goal is to solve a PCTA state planning problem which is defined as follows.

A *PCTA state plan planning problem* $Plan_{PCTA} = \langle A, s_0, SP_{goal}, h \rangle$ is comprised of a *parameterized concurrent timed automaton* A , *initial state* s_0 , *goal specification* SP_{goal} and *planning horizon* h . A is determinate and its transitions are specified in canonical form. s_0 is a (full) state assignment of A that occurs at e_0^+ , where e_0 is the earliest event. SP_{goal} is a state plan that includes start event e_0 and constrains the location and control variables of A . h is the maximum number of state changing events that may occur in the solution run.

An *end-state planning problem* is a PCTA state plan planning problem whose state plan is specified by $SP_{goal} = \langle c_g, c_s, [l, u] \rangle$ where c_g and c_s are state constraints, c_g holds from some event e_e to the end of the plan, c_s holds up to e_e , and the end goal is achieved within $e_e - e_0 \in [l, u]$.

3.2.2 Action Model

First we look at the definitions specific to the action model.

State Variable

Next, we formally define the notion of a state variable. Note that while a state variable’s assignment is a continuous time trajectory, in practice, the decision variables in the optimization encoding represent the state for a stage between two discrete time steps.

A *state variable* v is a variable whose assignment is a continuous timed trajectory $v(t) : \Re \rightarrow R(v)$, where $R(v)$ denotes the *range* of values that trajectory $v(t)$ can take on at a point in time.

Parameterized Timed Automaton

Now that we’ve defined the notion of a state variable, we introduce the formalism for a single automaton. For instance, the computer, connection and projector automata are all instances of a PTA.

A *parameterized timed automaton* A is described by octuple $\langle \mathcal{L}, \mathcal{C}, \mathcal{U}_I, \mathcal{U}_{I_C}, \mathcal{U}_P, \mathcal{U}_{I_P}, \mathcal{T}, \mathcal{R} \rangle$, where \mathcal{L} , \mathcal{C} , \mathcal{U}_I , \mathcal{U}_{I_C} , and \mathcal{U}_P are functions of time (e.g., *fluents*):

- $\mathcal{L} : \Re \rightarrow R(\mathcal{L})$ is the set of *location variables* of A . Each $l \in \mathcal{L}$ is a function of real-valued time with finite range $R(\mathcal{L}) = \{l_i\}$, representing locations over which this automaton transitions.
- $\mathcal{C} : \Re \rightarrow R(\mathcal{C})$ is the set of *clock variables* of A . Each $c \in \mathcal{C}$ is a function of real-valued time whose range is the non-negative reals ($R(c) = \Re_{\geq 0}$) and increases with time ($dc/dt = 1$).
- $\mathcal{U}_I : \Re \rightarrow R(\mathcal{U}_I)$ is the finite set of (exogenous) *input variables* of A . Each $u \in \mathcal{U}_I$ is a function of real-valued time with associated range $R(u) = \{u_i\}$. The elements of $R(u)$

are referred to as *commands*. When \perp is included in this range, it is called the *nullary command*, and indicates that no command is given.

- $\mathcal{U}_{I_C} : \mathfrak{R} \rightarrow R(\mathcal{U}_{I_C})$ is the set of *input clock variables* of A . Each $c \in \mathcal{U}_{I_C}$ is a function of real-valued time whose range is the non-negative reals ($R(c) = \mathfrak{R}_{\geq 0}$) and increases with time ($dc/dt = 1$).
- $\mathcal{U}_P : \mathfrak{R} \rightarrow R(\mathcal{U}_P)$ is the set of *clock parameters* of A . Each $u_p \in \mathcal{U}_P$ is a function of real-valued time and has an interval range $R(u_p) = [l, u] \subseteq \mathfrak{R}_{\geq 0}$.
- $\mathcal{U}_{I_P} : \mathfrak{R} \rightarrow R(\mathcal{U}_{I_P})$ is the set of *input clock parameters* of A . Each $u_p \in \mathcal{U}_{I_P}$ is a function of real-valued time and has an interval range $R(u_p) = [l, u] \subseteq \mathfrak{R}_{\geq 0}$,

where $\mathfrak{R}_{\geq 0} = \{x | x \in \mathfrak{R}, x \geq 0\}$.

$\mathcal{L}, \mathcal{C}, \mathcal{U}_I, \mathcal{U}_{I_C}, \mathcal{U}_P,$ and \mathcal{U}_{I_P} are aggregated to:

- $\mathcal{V} = \mathcal{L} \cup \mathcal{C}$ is the set of *state variables* of A . An assignment $\mathcal{V}(t)$ is called the *state* of automaton A at time t .
- $\mathcal{U} = \mathcal{U}_I \cup \mathcal{U}_{I_C} \cup \mathcal{U}_P \cup \mathcal{U}_{I_P}$ is the set of *control variables* of A , comprised of input variables, input clock variables, clock parameters, and input clock parameters. An assignment $\mathcal{U}(t)$ is called the *control assignment* of automaton A at time t .
- $\mathcal{W} = \mathcal{V} \cup \mathcal{U}$ is the set of *variables* of A , comprised of state and control variables of A .

PTA Transition and Reward

Given a definition of the PTA, we now describe several properties of the transitions between locations of the automaton and the rewards associated with those transitions.

Let A be a timed automaton, where $A = \langle \mathcal{L}, \mathcal{C}, \mathcal{U}_I, \mathcal{U}_{I_C}, \mathcal{U}_P, \mathcal{T}, \mathcal{R} \rangle$. Each transition $\tau \in \mathcal{T}$ of A is a quadruple $\tau = \langle l_s, l_t, g, c_r \rangle$ and each reward $\rho \in \mathcal{R}$ of A is a quadruple $\rho = \langle l_s, l_t, g, r \rangle$, where

- $l_s, l_t \in R(\mathcal{L})$ are *source* and *target locations*,
- $g : R(\mathcal{V}) \times R(\mathcal{U}) \rightarrow \{\top, \perp\}$ is the *guard relation*.
- *Clock reset* $c_r(\mathcal{C}', \mathcal{U}'_C)$ is a constraint over subsets of the clock variables and parameters, $\mathcal{C}' \subseteq \mathcal{C}$ and $\mathcal{U}'_C \subseteq \mathcal{U}_P$, respectively. $c_r(\mathcal{C}', \mathcal{U}'_C) = c_{rc}(\mathcal{C}')c_{ru}(\mathcal{U}'_C)$ is a composition of *clock variable reset* c_{rc} and *clock parameter reset* c_{ru} , where c_{rc} is an assignment to clock variables \mathcal{C}' and c_{ru} is an interval assignment to clock parameters \mathcal{U}'_C .
- *Reward value* $r \in \mathfrak{R}_{\geq 0}$ is a rate of reward per unit time.

We define several properties of transition τ and corresponding reward ρ :

- τ is an *idling* transition if $l_s(\tau) = l_t(\tau)$; otherwise, τ is a *state changing* transition.
- τ , respectively ρ , is *applicable at* t if $\mathcal{L}(t^-) = l_s$.

- τ , respectively ρ , is *enabled at t* if τ is applicable at t and if $g(\mathcal{C}(t^-), \mathcal{U}(t^-))$ is satisfied.
- When τ is enabled at t , automaton A *instantaneously transitions at t* from l_s to l_t and applies its clock reset c_r to its \mathcal{C}' and $\mathcal{U}'_{\mathcal{C}}$ at t^+ .
- τ is *satisfied at t* if three conditions hold, a) τ is enabled at t , b) $\mathcal{L}(t^+) = l_t$, and c) $\mathcal{C}(t^+) \wedge \mathcal{U}_{\mathcal{P}}(t^+) \models c_r$.
- Let ρ be applicable if and only if τ is applicable, then ρ is the *corresponding reward* to τ .

PTA Transition and Reward specification

For a transition to be satisfied, it must obey certain constraints over the starting state, the target state, the input variables, the clock variables, clock variable resets, and clock parameter resets. If this *transition specification* is satisfied then it results in the specified reward.

Let $\tau \in \mathcal{T}$ be a transition of parameterized timed automaton A , where $\tau = \langle l_s, l_t, g, c_r \rangle$, and let $\rho \in \mathcal{R}$ be its corresponding reward, where $\rho = \langle l_s, l_t, g, r \rangle$. A *transition and reward specification* for τ and ρ is a quintuple $\langle ss, ts, \varphi, cs, rs \rangle$, whose elements specify the corresponding elements of τ and ρ :

- $ss ::= (\mathcal{L} = l_s)$ and $ts ::= (\mathcal{L} = l_t)$ are *source* and *target specifications*, which assign *source* and *target* locations $l_s, l_t \in R(\mathcal{L})$, respectively.
- φ is the *guard specification*, which is a disjunction in which each disjunct φ_k consists of a *state condition* conjoined with a *clock condition*. Each *state condition* $\varphi_{s,k}$ specifies a relation over input variables $\mathcal{U}_{\mathcal{I}}$ of A .
Each *clock condition* $\varphi_{c,k}$ specifies a relation over clock variables $\mathcal{C} \cup \mathcal{U}_{I_C}$ and clock parameters $\mathcal{U}_{\mathcal{P}} \cup \mathcal{U}_{I_P}$.¹
- cs is the *clock reset specification*, which is a conjunction of a clock variable reset specification and a clock parameter reset specification. ctBurton restricts the clock variable reset specification to *False* for idling transitions and to assignments $c_i := 0$ for state changing transitions.
- $rs \in \mathfrak{R}_{\geq 0}$ is a *reward specification* such that rate of reward r is $r = rs$ when $l_s = l_t$ and $r = rs\delta(t)$ when $l_s \neq l_t$,

where $l \neq v$ denotes $\neg(l = v)$.

τ is satisfied at t if and only if $ss(t - \epsilon) \wedge \phi(t - \epsilon) \wedge ts(t + \epsilon) \wedge cs(t + \epsilon)$. Likewise, ρ evaluates to r at t if and only if $ss(t - \epsilon) \wedge \phi(t - \epsilon) \wedge ts(t + \epsilon) \wedge cs(t + \epsilon)$. τ and ρ are in *canonical form* if their state condition φ_s is in canonical form.

¹We choose l_i to be positive so that two or more state changing transitions can not occur instantaneously. We choose closed intervals for state changing transitions to ensure that state changes occur at well-defined events, and open intervals for idling transitions, to ensure that transitions are determinate.

Trace

In intuitive terms, the *trace* is a sequence of control inputs at the end of each stage (at an event) for each timed automaton.

Let \mathcal{V} be the state variables of A . Let $\mathcal{U} = \mathcal{U}_{\mathcal{I}} \cup \mathcal{U}_{\mathcal{P}} \cup \mathcal{U}_{\mathcal{I}_C} \cup \mathcal{U}_{\mathcal{I}_P}$ be a vector of control variables for timed automaton A , comprised of input variables $\mathcal{U}_{\mathcal{I}}$, clock parameters $\mathcal{U}_{\mathcal{P}}$, input clock variables $\mathcal{U}_{\mathcal{I}_C}$ and input clock parameters $\mathcal{U}_{\mathcal{I}_P}$. A (*valid*) *trace of \mathcal{U} over extent $(e_0, e_n) \subseteq \mathfrak{R}_{\geq 0}$* is a trajectory $\mathcal{U}(t)$ over (e_0, e_j) , where $e_j \leq e_n$ is the maximum event in (e_0, e_n) , such that \mathcal{U} and \mathcal{V} satisfy a transition $\tau_{ij} \in \mathcal{T}(A)$ for each $t \in (e_0, e_j)$, for some trajectory $\mathcal{V}(t)$ of A . $\mathcal{U}(t)$ is *complete* over (e_0, e_n) if $e_j = e_n$; otherwise, $\mathcal{U}(t)$ is *partial*.

Run and its reward

Stated simply, the *run* of a PCTA is the trajectory of the state variables over time given an initial state and the trace. In our optimization encoding, the state is parameterized as a sequence of states at stages between discrete consecutive events. The total reward for a run is the cumulative reward for all the idling and state changing transitions in the run which is formalized using an integral.

Let A be a parameterized timed automaton with control variables \mathcal{U} and state variables \mathcal{V} . Let $\mathcal{U}(t)$ be a trace of \mathcal{U} over $(e_0, e_n) \subseteq \mathfrak{R}_{\geq 0}$ and s_0 be the initial state of A at e_0^+ . A *run of A (equivalently \mathcal{V}) over (e_0, e_n)* , given s_0 and $\mathcal{U}(t)$, is a trajectory $\mathcal{V}(t)$ over (e_0, e_j) , where $e_j \leq e_n$ is the maximum event in (e_0, e_n) , such that a) $\mathcal{V}(e_0^+) = s_0$, b) for all $t \in (e_0, e_j)$, $\mathcal{V}(t) \in R(\mathcal{V})$ and c) \mathcal{U} and \mathcal{V} satisfy a transition $\tau_{ij} \in \mathcal{T}(A)$ for each $t \in (e_0, e_j)$. A run is *complete* if $t_k = t_n$; otherwise, it is *partial*.

Given a trace \mathcal{U} and its run \mathcal{V} of A , the reward of \mathcal{U} and \mathcal{R} over (e_0, e_n) is

$$\mathcal{R}(\mathcal{U}, \mathcal{V}, (e_0, e_n)) = \int_{t \in (e_0, e_n)} \mathcal{R}(\mathcal{V}(t - \epsilon), \mathcal{U}(t - \epsilon), \mathcal{L}(t + \epsilon)), \text{ as } \epsilon \rightarrow 0^+.$$

3.2.3 State Plan

Now that we've introduced all the key ideas relevant to the action model, we transition to those for the state plan.

Episode

An *episode* can loosely be thought of as an interval during the run that is constrained to satisfy a constraint over state variables and a temporal constraint between the start and end event of the episode. More formally,

An *episode ep* is a quintuple $\langle V, s, e, tc, sc \rangle$. The set of state variables V is the *scope of ep* . $s, e \in \mathfrak{R}$ are the *start and end events of ep* . Interval $[s, e]$ is the *extent of ep* . The extent is restricted by *temporal constraint tc* , called a duration constraint, where $s, e \in \text{scope}(tc)$. tc must imply that start s precedes or co-occurs with end e ($s \leq e$), and may further constrain the duration of the episode ($e - s$), also known as the temporal distance between s and e . V is restricted by a *state constraint sc* , for every $t \in (s, e)$.

State Plan

Intuitively, the state plan describes the evolution of the state of the system of concurrent automata over time and specifies the ways in which it should be constrained.

A *State Plan* SP is a quintuple $\langle \mathcal{V}, e_0, EV, EP, TC \rangle$ where:

- $\mathcal{V} = \{v_i\}$, called the *scope of SP* , is a set of state variables,
- e_0 is the *start event* of SP , denoting a non-negative, real-valued time point.
- $EV = \{e_i\}$ is the set of *events e_i of SP* , denoting non-negative, real-valued time points. EV is partitioned into *episode events EV_E* , which are start and end events of $ep_i \in EP$, and *reference events EV_R* , which are not episode events.
- $EP = \{ep_i\}$ is a set of *episodes of SP* , where the scope of each ep_i is a subset of \mathcal{V} and the start and end events of ep_i are in EV .
- $TC = \{tc_i\}$ is the set of *temporal constraints tc_i of SP* over events in EV . TC includes the constraint $e_0 \leq e_i$, for every event $e_i \in EV$, and the episode temporal constraints tc for every episode $ep_i \in EP$.

Valid Solution to $Plan_{PCTA}$

Finally, we say that a valid solution is defined as a trace such that executing it on the action model from a particular start state leads us to the desired goal state via a run that satisfies the state plan.

Given a PCTA planning problem $Plan_{PCTA} = \langle A, e_0, s_0, SP_{goal} \rangle$, a control plan $SP_{control}$ of $Plan_{PCTA}$ is a trace of A . $SP_{control}$ is a *valid solution* to $Plan_{PCTA}$ if the run of $SP_{control}$ on A , starting in s_0 at e_0 , satisfies state plan SP_{goal} .

3.3 Encoding - Decision Variables

Given the definitions, in this section we list the exact decision variables involved in implementing the CTBurton planner. The implementation uses the MiniZinc language.

Location variables

We define an array of location decision variables for each automaton of length **STAGES**. For instance, for storing the location of the connection automaton across stages one would write

```
enum ConnL = {connected, disconnected};  
array[STAGES] of var ConnL : connl;
```

Command variables

We define an array of command decision variables for each automaton of length **STAGES**. For instance, to store the commands for the projector automaton for all stages one would get

```
array[STAGES] of var ConnU : connu;
```

End State Accepting Automaton

We define three different decision variables for implementing the end stage accepting automaton.

1. **Automaton Location Trajectory** - This array of decision variables stores the state of the end state accepting automaton.

```
array[STAGES] of var endStateAcceptAutomatonL : endStateAcceptAutomatonTraj;
```

2. **Automaton Transition Trajectory** - This array of boolean decision variables stores which transition occurs in the end state accepting automaton during a particular stage.

```
array[endStateAcceptAutomatonTr, STAGES] of var bool: endStateAcceptAutomatonTrTraj;
```

3. **Acceptance stage** - This decision variable tracks the stage when the end state is reached.

```
var int: stageWhenAccepted;
```

Episode Accepting Automaton

We define a few decision variables for implementing the episode accepting automaton for each episode in the state plan.

1. **Automaton Location Trajectory** - This array of decision variables stores the state of the episode accepting automaton.

array[STAGES] of var epAcceptAutomatonL : epAcceptAutomatonTraj;

2. **Automaton Transition Trajectory** - This array of boolean decision variables stores which transition occurs in the episode accepting automaton during a particular stage.

array[epAcceptAutomatonTr, STAGES] of var bool: epAcceptAutomaton-TrTraj;

3. **Episode start event** - This decision variable tracks event when the episode starts.

var int : epAcceptStartEvent;

4. **Episode end event** - This decision variable tracks event when the episode ends.

var int : epAcceptEndEvent;

5. **Clock when episode starts** - This decision variable tracks the clock value when the episode starts.

var float : epAcceptClockWhenStart;

6. **Clock when episode ends** - This decision variable tracks the clock value when the episode ends.

var float : epAcceptClockWhenEnd;

7. **Episode Clock Parameter** - This decision variable store the value of the clock parameter for the episode.

var float : epAcceptClockParameter;

Temporal Constraint Accepting Automaton

We define a few decision variables for implementing the temporal constraint accepting automaton for each temporal constraint in the state plan. Note that event 1 does not have to occur before event 2.

1. **Automaton Location Trajectory** - This array of decision variables stores the state of the temporal constraint accepting automaton.

```
array[STAGES] of var tempAcceptAutomatonL : temporalAcceptConstraint-  
Traj;
```

2. **Automaton Transition Trajectory** - This array of boolean decision variables stores which transition occurs in the temporal constraint accepting automaton during a particular stage.

```
array[temporalAcceptAutomatonTr, STAGES] of var bool : temporalAccept-  
ConstraintTrTraj;
```

3. **Temporal constraint first event** - This decision variable tracks event when the episode starts.

```
var int : temporalAcceptConstraintevent1;
```

4. **Temporal constraint second event** - This decision variable tracks event when the episode ends.

```
var int : temporalAcceptConstraintevent2;
```

5. **Clock at first event** - This decision variable tracks the clock value when the first event is accepted.

```
var float : temporalAcceptConstraintevent1Clock;
```

6. **Clock at second event** - This decision variable tracks the clock value when the second event is accepted.

```
var float : temporalAcceptConstraintevent2Clock;
```

7. **Clock when temporal constraint is activated** - This decision variable stores the value of the clock when the temporal constraint is activated by either events being accepted.

```
var float : temporalAcceptConstraintClockWhenActivated;
```

8. **Clock when temporal constraint is accepted** - This decision variable store the value of the clock when the temporal constraint is accepted.

var float : temporalAcceptConstraintClockWhenAccepted;

9. **Temporal constraint clock parameter** - This decision variable store the value of the clock parameter for the temporal constraint.

var float : temporalAcceptConstraintClockParameter;

Event times

This array of decision variables stores the times at which the events occur.

array[STAGESp1] of var float : e;

Clocks for all stages for all automata

This variable stores the clock values for all automata in all stages.

array[AUTOMATA, STAGES] of var float : c;

Clock offsets for all stages

This variable stores the clock offsets for all stages for all automata.

array[AUTOMATA, STAGESp1] of var float : co;

Clock parameters for all stages

This variable stores the value of the clock parameters for all stages for all automata.

array[AUTOMATA, STAGES] of var float : cl;
array[AUTOMATA, STAGES] of var float : cu;

Open Interval Enablement (Eno)

This array of boolean decision variables keeps track of the the transitions that occur during the middle of each stage, i.e. during the interval.

array[TRANSITIONS, STAGES] of var bool: Eno;

End Interval Enablement (Ene)

This array of boolean decision variables keeps track of the transitions that occur at the end of each stage.

array[TRANSITIONS, STAGES] of var bool: Ene;

3.4 Encoding - Constraints

In this section, we define the constraints over the decision variables defined in the previous section in order to enforce properties of the CTBurton planner.

3.4.1 Event constraints

All variables that store event numbers are constrained to hold integer values between $[0, \dots, h]$ (both inclusive). For instance,

constraint epAcceptStartEvent $\geq 0 \wedge$ epAcceptStartEvent $\leq h$;

3.4.2 Clock or clock parameters

All variables that store clock variables or clock parameters are float variables that are bounded in the range $[0, \dots, Fupper]$. For instance,

constraint epAcceptClockWhenStart $\geq 0 \wedge$ epAcceptClockWhenStart $\leq Fupper$;

3.4.3 Ordering of events

The timestamps for events should be monotonically increasing with the event index.

constraint forall(i in STAGES)(e[i] < e[i+1]);

3.4.4 Clock variable and clock offset consistency

The clock variable for a given stage is the difference between the time stamp of the end event for that stage and the clock offset for the stage.

constraint forall(a in AUTOMATA, i in STAGESm1)(c[a,i] = e[i+1] - co[a,i])

3.4.5 Transitions within a stage

State changing transitions

State changing transitions can't occur within a stage by construction since the state has to remain the same during a particular stage. This is encoded by requiring the boolean **Eno** decision variables to be false for every state changing transition τ .

$$\neg En_o(\tau, i). \tag{3.1}$$

One instance of the minizinc code for an automaton for state changing transition 1 will look like

constraint forall(i in STAGES)(not Eno[1,i]);

Idling transitions

Idling transitions can occur within a stage but they do not involve clock variable or clock parameter resets which can be written as the following constraint.

$$En_o(\tau, i) \implies ss(\mathcal{L}_i) \wedge \bigvee_k (\varphi_{s,k}(\mathcal{L}_i, \mathcal{U}_{I_i}) \wedge \varphi_{co,k}(\mathcal{C}_i, \mathcal{U}_{I_{C_i}}, \mathcal{U}_{P_i}, \mathcal{U}_{I_{P_i}})). \quad (3.2)$$

where ss refers to the system location satisfying the start state for that transition. The target state constraint is not necessary since the state does not change during an idling transition. The clock variables $\varphi_{c,k}$ are required to be within a certain range $[0, u_{k,j}]$.

$$\varphi_{c,k} ::= \bigwedge_j c_j \in [0, u_{k,j}],$$

and the end event timestamp and the clock offsets must also satisfy the range $[0, u_{k,j}]$.

$$\varphi_{co,k}(\mathcal{C}_i, \mathcal{U}_{I_{C_i}}, \mathcal{U}_{P_i}, \mathcal{U}_{I_{P_i}}) ::= \bigwedge_j (e_i^+ - c_{i,j}) \in [0, u_{k,j}]$$

One example of the minizinc encoding with idling transition 5 for the computer automaton looks like

```
constraint forall(i in STAGES)(Eno[5,i] -> compl[i]==compOn /\ openIdlingClock(c,
cl, cu, e, co, comp, i));
```

which uses the following primitives/predicates

```
predicate lessThanUpper(var float: eStart, var float: eEnd, var float: co, var
float: u) = (eEnd - co <= u);
```

```
predicate openIdlingClock(array[Automata,int] of var float: c, array[Automata,int]
of var float: l, array[Automata,int] of var float: u, array[int] of var float: e, ar-
ray[Automata,int] of var float: co, Automata: a, int: i) = lessThanUpper(e[i],
e[i+1], co[a,i], u[a,i]);
```

3.4.6 Transitions between stages and associated resets

The transitions at the end of stages are encoded by constraining the **Ene** decision variable for each state and idling transition for each stage. Mathematically, this constraint is written as the equation below. This accounts for the starting location in the current stage and the target stage in the next location. In addition to the clock variable and clock parameter constraints this also accounts for clock variable resets and the clock parameter reset when an automaton transitions from one location to another.

$$En_e(\tau, i) \implies ss(\mathcal{L}_i) \wedge \bigvee_k (\varphi_{s,k}(\mathcal{L}_i, \mathcal{U}_{I_i}) \wedge \varphi_{ce,k}(\mathcal{C}_i, \mathcal{U}_{I_{C_i}}, \mathcal{U}_{P_i}, \mathcal{U}_{I_{P_i}})) \wedge ts(\mathcal{L}_{i+1}) \wedge cvs(\mathcal{C}_{i+1}) \wedge cps(\mathcal{U}_{i+1}). \quad (3.3)$$

where the clock variable and clock parameter constraints are similar to the constraints for transitions within a stage with a change that the lower bound is a parameter $l_{k,j}$ for state changing transitions and 0 for idling transitions.

For idling transitions,

$$\varphi_{ce,k}(\mathcal{C}_i, \mathcal{U}_{IC_i}, \mathcal{U}_{P_i}, \mathcal{U}_{IP_i}) ::= \bigwedge_j (e_i^+ - c_{ij}) \in [0, u_{k,j}].$$

For state changing transitions,

$$\varphi_{ce,k}(\mathcal{C}_i, \mathcal{U}_{IC_i}, \mathcal{U}_{P_i}, \mathcal{U}_{IP_i}) ::= \bigwedge_j (e_i^+ - c_{i,j}) \in [l_{k,j}, u_{k,j}]$$

If the clock variable is reset at a particular end event e_i^+ , then the clock offset $c_{i+1,j}$ for the next stage is the end event time i.e. the time at which the clock is reset.

$$cvs(\mathcal{C}_{i+1}) ::= \forall (c_j := 0) \in cvs.En_e(\tau_k, i) \rightarrow c_{i+1,j} = e_{i+1}.$$

Similar for clock parameter resets, the clock parameter is constrained to be between some lower bound (rather than zero as in case of the clock) and upper bound.

$$cps(\mathcal{U}_{i+1}) ::= \forall (p_j \in [l_j, u_j]) \in cps.En_e(\tau_k, i) \rightarrow (p_{i+1,j} \in [l_j, u_j]).$$

An example of the minizinc code for the computer automaton for the state changing transition 1 –

```
constraint forall(i in STAGESm1)(Ene[1,i] ->
compl[i]==compOn /\ compl[i+1]==compShutdown /\
compu[i]==compTurnOff /\
closeChangeClock(c, cl, cu, e, co, comp, i) /\
cvs(e, co, comp, i) /\ cps(co, cl, cu, comp, i));
```

3.4.7 Making clock variables and parameters persist

We must separately handle scenarios where the clock variables and parameters are not reset at the end of a stage, in which case those values need to be maintained and propagated into the next stage.

We know that a clock variable is not reset if none of the state changing transitions occur. This encoded in the form of the constraint below

$$maintain(\mathcal{C}_{i+1}) ::= \forall c_j \in \mathcal{C}. \left(\bigwedge_{\tau_k \in T(c_j)} \neg En_e(\tau_k, i) \right) \rightarrow c_{i+1,j} \equiv c_{i,j}. \quad (3.4)$$

The constraint for clock parameters is similar

$$maintain(\mathcal{U}_{i+1}) ::= \forall p_j \in \mathcal{U}. \left(\bigwedge_{\tau_k \in T(p_j)} \neg En_E(\tau_k, i) \right) \rightarrow p_{i+1,j} \equiv p_{i,j}. \quad (3.5)$$

An example of the minizinc encoding for the computer automaton would be

```

constraint forall(i in STAGESm1)
  ((forall([(not Ene[t, i]) | t in TRANSITIONS_compc_reset])) ->
   (co[comp,i]==co[comp,i+1]));

```

3.4.8 Constraints that ensure determinate transitions

The CTBurton planner requires that the transitions in the automata be deterministic, and that only one transition can occur at a given time. This is enforced in two parts by constraining that only one transition is enabled during a stage, and that only one transition is enabled at the end of a stage for an automaton A_p .

$$\begin{aligned}
 \text{Determinate}(A_p, n) ::= & \\
 & \text{Determinate}_o(A_p, n) \wedge (\forall 1 \leq i < n. \text{Determinate}_o(A_p, i) \wedge \text{Determinate}_e(A_p, i)). \quad (3.6)
 \end{aligned}$$

We first define two sets

$$\begin{aligned}
 S_o(A_p, i) ::= & \{En_o(\tau_k, i) \mid \tau_k \in \mathcal{T}_I(A_p)\} \\
 S_e(A_p, i) ::= & \{En_e(\tau_k, i) \mid \tau_k \in \mathcal{T}(A_p)\},
 \end{aligned}$$

and consequently enforce that only of the elements in this set must be true.

$$\begin{aligned}
 \text{Determinate}_o(A_p, i) ::= & \text{OneOf}(S_o(A_p, i)) \\
 \text{Determinate}_e(A_p, i) ::= & \text{OneOf}(S_e(A_p, i)),
 \end{aligned}$$

Finally, we enforce this constraint across all the automata.

$$\text{Determinate}(A, n) ::= \bigwedge_{A_p \in \mathcal{A}(A)} \text{Determinate}(A_p, n), \quad (3.7)$$

The equivalent minizinc code for the computer automaton takes the form

```

constraint forall(i in STAGES)(sum([(Eno[t, i] | t in TRANSITIONS_comp_idle]
= 1);

```

```

constraint forall(i in STAGES)(sum([(Ene[t, i] | t in (TRANSITIONS_comp_idle
union TRANSITIONS_compc_reset)]) = 1);

```

3.4.9 Constraints that ensure concise runs

We introduce a set of constraints to ensure conciseness of plans so that if the time horizon h is longer than is necessary, the CTBurton planner doesn't allow the automata to idle if they haven't reached the goal state.

This is enforced by ensuring that either atleast one automaton should have a state changing transition in any given stage or the goal state accepting automaton must be in the accepting location.

$$\forall i, \quad 0 \leq i < h - 1 \quad \text{concise}(i),$$

where

$$\text{concise}(i) ::= \left(\bigvee_{\tau_k \in T_{sc}(A)} \text{En}_e(\tau_k, i) \right) \text{ xor } \text{end_state_accepted}$$

The minizinc for implementing this looks as follows -

```
constraint forall(i in STAGESm1)
  (exists([Ene[t,i] | t in TRANSITIONS_state_changing]) xor
  endStateAcceptAutomatonTraj[i]==accepted);
```

3.5 Encoding - State Plan

In this section, we look at the encoding for the state plan.

3.5.1 Encoding Initial State

Encoding the initial state is straightforward and can be accomplished by constraining the location of the automata at time step 0. We also ensure that the end state accepting automaton's location at time step 0 is set to waiting.

An example for the presentation scenario would be

```
constraint compl[0]==compOff;
constraint connl[0]==disconnected;
constraint projl[0]==projOff;
constraint endStateAcceptAutomatonTraj[0]==waiting;
```

3.5.2 End State Accepting Automaton

For the plan to be valid, there must be atleast one stage in which the end state accepting automaton is in the 'accepted' state.

The minizinc is written in the form

```
constraint exists(i in STAGES)(endStateAcceptAutomatonTraj[i]==accepted);
```

The end state accepting automaton can be described using a set of locations and the guard conditions for transitioning between those locations based on a maintenance constraint $mc(\mathcal{V})$ and a goal constraint $gc(\mathcal{V})$.

- **Locations** - Waiting, Accepted and Failed
- **Transitions** -
 - Idle on Waiting when $mc(\mathcal{V}) \wedge \neg gc(\mathcal{V})$;
 - Waiting to Accepted when $gc(\mathcal{V})$;
 - Waiting to Failed when $\neg mc(\mathcal{V}) \wedge \neg gc(\mathcal{V})$;
 - Idle on Accepted always (\top), and
 - Idle on Failed always (\top).

The constraints for enforcing the transitions of the end state accepting automaton are similar to the other automata. Please check the code on github for the full treatment. Finally we enforce one last constraint that only one of those transitions can occur in a given stage.

Finally we also enforce that the goal state accepting automaton cannot enter it's accepted state unless the episode accepting, temporal constraint accepting and the event accepting automaton have also been accepted.

3.5.3 Episode Accepting Automaton

An episode accepting automaton is similar to an end state accepting automaton with a few differences. An episode accepting automaton also has a clock variable c and a clock parameter δ . The state constraint $sc(\mathcal{V})$ must hold for the entire episode. The clock is reset when the automata enters the active location.

- **Locations** - Waiting, Active, Accepted, Failed.
- **Transitions** -
 - idle on Waiting when $\neg \text{Accepting}(s)$,
 - Waiting to Active when $\text{Accepting}(s)$; reset $c = 0$ and $\delta \in [l, u]$,
 - idle on Active when $sc(V)$ and $c < \delta$,
 - Active to Accepted when $sc(V)$ and $\delta \leq c$,
 - Active to Failed when $\neg sc(V)$,
 - idle on Accepted always (\top), and
 - idle on Failed always (\top).

If the state at time step 0 satisfies the accepting constraint for the episode, the automaton starts in the ‘Active’ location, or else it starts in the ‘Waiting’ location. Similar to the prior automata, we enforce that only one transition can occur at the end of each stage. This translates to the following minizinc code

```
constraint (projl[0]==projOn)->(epAcceptAutomatonTraj[0]==epActive);
constraint (projl[0]!=projOn)->(epAcceptAutomatonTraj[0]==epWaiting);
```

Finally, we ensure that the duration of the episode is equal to the clock parameter.

```
constraint (epAcceptClockWhenEnd - epAcceptClockWhenStart)==epAcceptClockParameter;
```

3.5.4 Temporal Constraint Accepting Automaton

A temporal constraint accepting automaton is similar to an episode accepting automaton but without the state constraint and more flexibility in the ordering of events.

A temporal constraint accepting automaton does not enforce a strict ordering over the events so either event can get accepted first and the automaton will transition from waiting to active. Once active, the automaton will transition to accepted once the clock variable is greater than or equal to the clock parameter.

Depending on whether the initial state satisfies the accepting constraints for either events, the automaton could start in the Active state or the Waiting state. If both accepting constraints are satisfied simultaneously by the initial state then the automaton could start in the Accepted state. Finally, we ensure that only one transition can occur at a time and that the difference between the end event clock and the start event clock is the clock parameter for the automaton.

We direct the reader to the full minizinc implementation for a thorough treatment.

3.5.5 Event Accepting Automaton

Now that we've already encoded the episode accepting and the temporal constraint accepting automaton, the final automata are the event accepting automaton which are responsible for synchronizing the start and events of the various temporal constraint and episode accepting automata together.

The event accepting automaton constraints ensure that for a given event that is a shared starting or ending event across constraints, all the constraints are accepted or all the constraints are waiting or active. If some of the constraints associated with that event are accepting and others are not, the automaton transitions to the failed location.

- Locations: Waiting, Accepted, and Failed;
- Transitions:
 - Idle on Waiting when $\bigwedge_{ep \in EP(e_i)} \neg \text{Accepting}(ep) \wedge \bigwedge_{tc \in TC(e_i)} (\text{Active}(tc) \wedge \neg \text{Accepting}(tc) \vee \text{Waiting}(tc))$;
 - Waiting to Accepted when $\bigwedge_{ep \in EP(e_i)} \text{Accepting}(ep) \wedge \bigwedge_{tc \in TC(e_i)} (\text{Active}(tc) \wedge \text{Accepting}(tc) \vee \text{Waiting}(tc))$;
 - Waiting to Failed when $\left[\left(\bigvee_{ep \in EP(e_i)} \text{Accepting}(ep) \right) \vee \left(\bigvee_{tc \in TC(e_i)} \text{Active}(tc) \wedge \text{Accepting}(tc) \right) \right] \wedge \left[\left(\bigvee_{ep \in EP(e_i)} \neg \text{Accepting}(ep) \right) \vee \left(\bigvee_{tc \in TC(e_i)} \text{Active}(tc) \wedge \neg \text{Accepting}(tc) \right) \right]$;
 - Idle on Accepted always (\top), and
 - Idle on Failed always (\top).

3.6 Encoding - Objective/Cost Function

In the implementation for this thesis we use a reward function without dense rewards which do not account for the cost of idling transitions, and only consider the cost of state transitions. The total cost is the sum over all stages over all automata of the cost associated with each transition. Mathematically, this gives us a cost or reward function of,

$$R(Tr_i(A)) = \sum_{0 < i < h} \sum_{A_j \in \mathcal{A}} \left[\sum_{\tau_k \in \mathcal{C}_I(A_i)} c_p(En_e(\tau_k, i)) r(\rho_k) \right]. \quad (3.8)$$

3.6.1 Optimization Problem Formulation

The encoding we present in the section above can be instantiated in slightly different ways depending on the nature of the constraints and the objective function. The encoding we presented above is a deterministic mixed logical temporal program (DMLTP). It's possible to turn this into a temporal optimal satisfiability problem if some conditions are satisfied.

Mixed Logic Temporal Program

A mixed state logic temporal problem (MLTP) is a constraint optimization problem whose constraints are propositional logic formula with propositions that include assignments to finite domain state variables and simple temporal constraints over real-valued events.

A *Mixed Logic Temporal Problem (MLTP)* is denoted by a quintuple $\langle \mathcal{P}_A, \mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{S} \rangle$. \mathcal{P}_A is a set of atomic propositions, \mathcal{V} is a set of finite domain variables with domains \mathcal{D}_V , and \mathcal{E} is a set of non-negative, real-valued events. $\mathcal{P} \equiv \mathcal{P}_A \cup \mathcal{P}_V \cup \mathcal{P}_E$ is a set of propositions, where $\mathcal{P}_V \equiv \{x = v | x \in \mathcal{V}, v \in \mathcal{D}_x\}$ is the set of all single variable assignments to \mathcal{V} and \mathcal{P}_E is the set of all temporal constraints over events \mathcal{E} . \mathcal{F} is a linear objective function that maps assignments over \mathcal{P}_A , \mathcal{V} and \mathcal{E} to the reals, that is, $\mathcal{F}(\mathcal{P}_A, \mathcal{V}, \mathcal{E}) = \mathcal{F}_P(\mathcal{P}_A) + \mathcal{F}_V(\mathcal{V}) + \mathcal{F}_E(\mathcal{E})$. \mathcal{S} is a set of *mixed state logic temporal formula* over \mathcal{P} , which are propositional logic formula, using standard connectives $\{\vee, \wedge, \neg, \implies, \iff\}$, whose propositions are drawn from a) atomic propositions \mathcal{P}_A , b) finite domain variable assignments \mathcal{P}_V , and c) temporal constraints \mathcal{P}_E .

An MLTP is parametric if it includes parametric temporal constraints. A parametric temporal constraint is a temporal constraint whose interval bounds $[l, u]$ may be events as well as constants ($l, u \in \mathcal{E} \cup \mathbb{R}$).

3.6.2 Temporal Optimal Satisfiability (TOP) problem

A *Temporal Optimal Satisfiability Problem* is a mixed logic temporal problem with three restrictions. First, its linear objective function \mathcal{F} is over atomic propositions \mathcal{P} and finite domain variables \mathcal{V} , but not real-valued events \mathcal{E} , that is, $\mathcal{F}_E(\mathcal{E}) = 0$. A candidate is an assignment to \mathcal{P} and \mathcal{V} such that there exists an assignment to \mathcal{E} that satisfies constraints \mathcal{S} . Third, temporal constraints \mathcal{P}_E are non-parametric, that is, l and u of each interval bound of \mathcal{P}_E is restricted to constants.

3.6.3 Deterministic Mixed Logic Temporal Problem

A *Deterministic Mixed Logic Temporal Problem (DMLTP)* is a MLTP whose events \mathcal{E} and temporal constraints \mathcal{P}_E form a (deterministic) simple temporal network $\langle \mathcal{E}, \mathcal{P}_E \rangle$.

3.7 Implementation Details

This section describes some of the lower level details necessary to write a functioning minizinc encoding.

Upper bound on float variables

We introduce a float variable *Fupper* that is used as an upper bound for all other float decision variables in the encoding due to the gecode solver's inability to handle unbounded floats correctly.

h

We also introduce another hyperparameter called *h* which determines the fixed preset number of events that the final state and control trajectory will have.

STAGES, STAGESm1, STAGESp1

The interval between every consecutive pair of events is defined as a stage. As a result the number of stages is 1 less than the time horizon *h*.

Chapter 4

Natural Language to State Plans

Given a state plan, we've looked at the optimization encoding for the CTBurton encoding which uses the action model and the state plan to compute the full state and control trajectory. Next we turn towards the problem of converting a user command in natural language to a state plan (and choose an appropriate representation).

We hypothesize that a large language model can reliably translate a command in natural language to a state plan because the state plan uses concepts that correspond directly with linguistic counter parts, that is episodes of behavior and common linguistic temporal constraints. Furthermore, we argue that this is more reliable than translating to a grounded plan, or to a constraint program (what you start with).

Theoretically, one might argue that one could try to get the LLM to learn a direct mapping between natural language and the Minizinc CTBurton encoding. However, the full encoding can run to up to many hundreds (for simpler examples) to thousands of lines (for more complex problems) which significantly increases the chances of the LLM making a mistake. As much as possible, we would like for the translation from natural language to the encoding to be as deterministic as possible which is accomplished by a python program that *compiles* an intermediate form of the **state plan** (a JSON file) to the minizinc encoding.

We accomplish this by using an LLM to generate this intermediate JSON form which is then compiled to a Minizinc encoding.

The first key insight was that the structure of the natural language input would map more naturally to a state plan as opposed to other representations (like temporal logic). It has also been established that the performance of large language models suffers on longer horizon tasks and the output eventually becomes incoherent or drifts away from the original task. The other key idea here is that instead of generating the entire minizinc encoding using a large language model, we only generate a JSON file which has much fewer lines (on the order of less than 20 to 100) and contains only a handful of strings that represent standard minizinc code.

4.1 JSON specifications

Here we'll look at the specification for the various JSON files that are provided as input to our method.

4.1.1 System config

The *system_config.json* is independent of the user input and the scenario and it specifies the information relevant to the various accepting automata introduced in the previous section. This file is internal to the ctBurton implementation and the user will not have to interact with it. We outline it here for completeness.

```
1 {
2     "transition_offset" : 0,
3     "end_state_accepting_automaton": {
4         "locations": ["waiting", "accepted", "failed"],
5         "transitions": [28, 29, 30, 31, 32]
6     },
7     "episode_accepting_automaton": {
8         "locations": ["epWaiting", "epActive", "epAccepted", "
9             epFailed"],
10        "transitions": [33, 34, 35, 36, 37, 38, 39]
11    },
12    "temporal_constraint_accepting_automaton": {
13        "locations": ["tempWaiting", "tempActive", "tempAccepted", "
14            tempFailed"],
15        "transitions": [41, 42, 43, 44, 45, 46]
16    }
17 }
```

4.1.2 Experiment Config

The *experiment_config.json* specifies details about the specific scenario, the time horizon, the types of automata, the number of instantiated automata, the number and types of transitions, and other details specific to the automata (such as the locations). For a given action model, this file will *not* change. The schema for the experiment config looks as follows

```
-
1 {
2     "state_plan_file_path": "<path_to_json>.json",
3     "Fupper": 25000,
4     "h": <time_horizon>,
5     "automata_types": [
6         "automaton_type_1",
7         "automaton_type_2",
8         "automaton_type_3"
9     ],
10    "instantiated_automata": [
11        "automaton_1",
12        "automaton_2",
13        "automaton_3"
14    ],
15    "tr" : <num_transitions>,
```

```

16 "transition_types" : ["idle", "reset"],
17 "transitions":
18   {
19     "idle":
20     {
21       "automaton_1": [
22         <idle_transition_1>,
23         ...
24       ],
25       ...
26     },
27     "reset":
28     {
29       "automaton_1": [
30         <reset_transition_1>,
31         ...
32       ],
33       ...
34     },
35     "state_changing": [
36       <state_changing_transition_1>,
37       ...
38     ]
39   },
40 "automata_locations":
41 {
42   "<automaton_1>" :
43   [
44     "location_1", "location_2"
45   ],
46   "<automaton_2>" :
47   ...
48 },
49 "automata_actions":
50 {
51   "<automaton_1>" :
52   [
53     "action_1", "action_2"
54   ],
55   "<automaton_2>" :
56   ...
57 }
58 }

```

4.1.3 State Plan

The state plan is an encoding of the desired task, *translated from the user's natural language input*. The general schema for this looks as follows

```
1 {
2   "initial_state":
3   {
4     "automaton_1" : "initial_location",
5     ...
6   },
7   "end_state":
8   {
9     "maintenance_constraint" : "<constraint_minizinc_string>",
10    "goal_constraint" : "constraint_minizinc_string"
11  },
12  "constraints":
13  {
14    "episodes":
15    {
16      "episode_1":
17      {
18        "accepting_constraint": "constraint_minizinc_string",
19        "state_constraint" : "constraint_minizinc_string",
20        "lower_bound": <lower_bound>,
21        "upper_bound": <upper_bound>
22      }
23      ...
24    },
25
26    "temporal":
27    {
28      "temporal_constraint_1":
29      {
30        "accepting_event1_constraint": "
31          constraint_minizinc_string",
32        "accepting_event2_constraint": "
33          constraint_minizinc_string",
34        "lower_bound": <lower_bound>,
35        "upper_bound": <upper_bound>
36      },
37      ...
38    },
39
40    "event":
41    {
```

4.2 Prompting approaches

Now we look at the two different prompting methods that were used in this thesis to generate the state plan JSON outlined in the previous section. The exact context necessary for the prompts is included in the gitlab repository due to its length, but we describe its key characteristics here. We will demonstrate the performance of both these methods directly in the next chapter on experimental results.

Let's consider a pedagogical example for both the LLM prompting methods where the model is expected to provide the correct answer to the question as the output.

4.2.1 Direct JSON generation

Pedagogical Example

Here is my problem: "Michael is a 31 year old man from America. He is at that really famous museum in France looking at its most famous painting. However, the artist who made this painting just makes Michael think of his favourite cartoon character from his childhood. What was the country of origin of the thing that the cartoon character usually holds in his hand?" *Solve this:*

In response to this, ChatGPT returns the following output where it's unable to provide the most likely answer to the question.

Since the problem doesn't provide specific details about the cartoon character, we can't directly identify what the character holds in their hand. To proceed you would need more information about the cartoon character in question. Without that information, it's impossible to determine the country of origin of the item the character holds.

Usage with CTBurton

Similar to the previous example where we ask the model to generate a solution in one go, when using the LLM to generate a state plan JSON, we ask the large language model to directly generate the *entire* state plan JSON file in one shot. This is the most straightforward method since it does not involve any intermediate steps. However, as is evident, it is also more susceptible to mistakes or "hallucinations" in the output as we find in our experiments.

4.2.2 Chain-of-Thought (CoT) prompting

The natural language processing community has found that using a step-by-step method can often lead to better overall outputs (albeit taking more steps) compared the the direct

output generation method. In this spirit we translate an NL statement to a state plan using the following series of steps.

The core principle of chain-of-thought prompting [37] is to make a large language model think step-by-step similar to the way humans think by breaking down a larger problem into smaller sub-problems which are easier to solve, and then chaining the solutions together to solve the full problem.

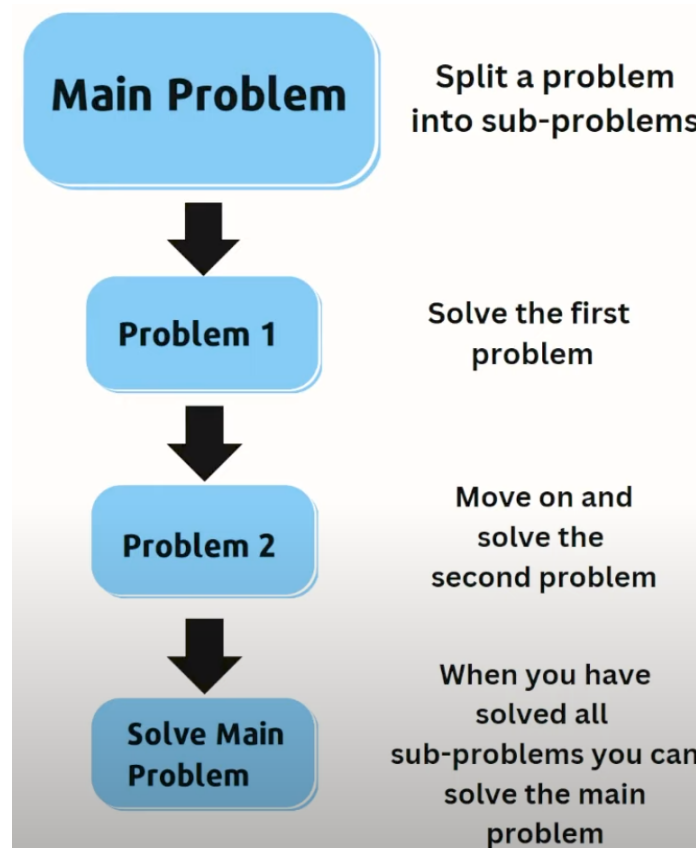


Figure 4.1: Chain of thought prompting relies on breaking down a larger problem into smaller subproblems.

Pedagogical example

We slightly alter the prompt to the following

Here is my problem: "Michael is a 31 year old man from America. He is at that really famous museum in France looking at its most famous painting. However, the artist who made this painting just makes Michael think of his favourite cartoon character from his childhood. What was the country of origin of the thing that the cartoon character usually holds in his hand?" *Without solving this problem just yet, think through this carefully and list systematically and in detail all the problems in the riddle that need to be solved before we can arrive at the correct answer.*

In response to which the LLM provides the following list of smaller sub-questions.

Let's break down the problems or elements that need to be addressed in the given riddle before arriving at the correct answer:

1. Identify Michael's location
2. Identify the most famous painting
3. Identify the artist of the most famous painting.
4. Determine Michael's favorite cartoon character.
5. Identify the cartoon character's usual object.
6. Determine the country of origin of the object.

We then prompt the LLM **again** repeatedly and ask it to give us the *most likely* answer to each of the questions 1 through 6 sequentially. This results in the correct output.

Usage with CTBurton

In this method, we ask the large language model to withhold generating the entire state plan JSON file. Instead, we ask the model to identify and extract the relevant information from the input text. Specifically, we ask the language model to first list all the episode and temporal constraints it can identify as well as the initial and end state.

Following that, as we will see in the next section, we sequentially request the large language model to map it's own intermediate text output with the information it has extracted onto the final state plan JSON file.

Chapter 5

Experimental Results

We analyze the performance of our method across a variety of different motivating examples introduced in the introductory chapter. We also analyze other properties of the method such as the scalability of the method with the number of agents in the environment, and the number of transitions in a given action model, number of constraints by measuring the impact on solving time. We also test the performance of the method across a variety of different natural language inputs and highlight some of the shortcomings of the method.

The Minizinc encoding is solved using the GeCode solver which is run in an anytime fashion and will sometimes return a suboptimal output and continue looking for the optimal solution to the optimal satisfiability problem. The optimal output decision variables from CTBurton for the various input prompts are included in the Appendix.

5.1 Computer-Connection-Projector

5.1.1 Initial in-context learning prompt

The prompt looks as follows -

```
1   Your goal is take a natural language input from the user
   describing the behavior of a certain set of agents and convert
   it into a state plan json file that is consistent with that
   input. This will then subsequently be processed by a
   generative planner called CTBurton which will use the state
   plan in conjunction with the action model to output a full
   state-control trajectory.
2
3   The system_config.json specifies transitions and locations for
   the accepting automatons and other details which do not
   change. The experiment_config.json specifies details relating
   to the automatons, the time horizon and a link to the
   state_plan.json. For a fixed action model, this file should
   also not change. Your primary goal is to output a
   state_plan.json file corresponding to the given natural
   language input.
```

4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

I've provided you with the code that performs the translation from the json files to the minizinc encoding which is used by CTBurton. I've also provided you with the system_config.json and experiment_config.json file. Moreover, you're provided with pairs of natural language inputs and the corresponding state_plan.json file. Your goal is to take new inputs from the user, and output the correct state_plan.json file.

system_config.json

```
{  
"transition_offset" : 0,  
... <omitted rest of the file>  
}
```

experiment_config.json

```
{  
"state_plan_file_path": "state_plan.json",  
"Fupper": 25000,  
... <omitted rest of the file>  
}
```

Natural language Input -

The computer and projector start in the off state, and the connection is disconnected. The projector should turn on and stay on for 1800 seconds, following which everything should turn off within 600 seconds.

state_plan.json

```
{  
  "initial_state":  
  {  
    "comp" : "compOff",  
    "conn" : "disconnected",  
    "proj" : "projOff"  
  },  
  "end_state":  
  {  
    "maintenance_constraint" : "true",  
    "goal_constraint" : "compl[i+1]==compOff /\ \ connl[i+1]==  
                        disconnected /\ \ projl[i+1]==projOff"  
  },  
  "constraints":
```

```

41     {
42         "episodes":
43         {
44             "Aasdas":
45             {
46                 "accepting_constraint": "proj1[i]==proj0n",
47                 "state_constraint" : "proj1[i]==proj0n",
48                 "lower_bound": 1800,
49                 "upper_bound": 1800
50             }
51         },
52         "temporal":
53         {
54             "last600":
55             {
56                 "accepting_event1_constraint": "proj1[i]==proj0n
57                 /\ \ proj1[i+1]==projConfirm",
58                 "accepting_event2_constraint": "proj1[i+1]==
59                 proj0ff /\ \ conn1[i+1]==disconnected /\ \ compl[
60                 i+1]=comp0ff /\ \
61                 temporalAcceptConstraint_Traj_last600[i]==
62                 tempActive",
63                 "lower_bound": 600,
64                 "upper_bound": 600
65             }
66         },
67         "event":
68         {
69             }
70     }
71
72 run.py #This is the file for running a state plan with CTBurton.
73
74 ... <omitted file content>
75
76 PyCTBurton.py # This file is called by run.py which translates
77 the state plan json to the minizinc encoding.
78
79 ... <omitted file content>

```

Finally, depending on the exact prompting method used we have two different variations for the end of the prompt.

5.1.2 Direct JSON generation

```
1 After processing all this, wait for the user to provide a new
   natural language text input, and once it s provided output
   the corresponding state_plan.json.
```

5.1.3 Chain-of-thought prompting

```
1 After processing all this, wait for the user to provide a new
   natural language text input, and once it s provided,
   identify and extract the initial state of the automata, the
   specified end state, the episodes and the temporal constraints
   without directly generating the state_plan.json.
```

5.1.4 Test prompts

Test prompt 1 The computer and projector start in the off state, and the connection is disconnected. The projector should turn on and stay on for 1800 seconds, following which everything should turn off within 600 seconds. In addition, the projector should turn on within 60 seconds.

Test prompt 2 The computer and projector start in the off state, and the connection is disconnected. The projector should turn on and stay on for 200 seconds, the computer should stay on for 300 seconds, following which everything should turn off within 600 seconds.

Direct JSON generation

Test prompt 1 Here's the output from the large language model for the test prompt 1.

```
1 {
2   "initial_state": {
3     "comp": "compOff",
4     "conn": "disconnected",
5     "proj": "projOff"
6   },
7   "end_state": {
8     "maintenance_constraint": "true",
9     "goal_constraint": "compl[i+1]==compOff /\ \ connl[i+1]==
10    disconnected /\ \ projl[i+1]==projOff"
11  },
12  "constraints": {
13    "episodes": {
14      "ProjOn": {
15        "accepting_constraint": "projl[i]==projOn",
16        "state_constraint": "projl[i]==projOn",
17        "lower_bound": 1800,
18        "upper_bound": 1800
19      }
20    }
21  }
22 }
```

```

19     },
20     "temporal": {
21         "last600": {
22             "accepting_event1_constraint": "proj1[i]==projOn /\
proj1[i+1]==projConfirm",
23             "accepting_event2_constraint": "proj1[i+1]==projOff
/\
conn1[i+1]==disconnected /\
compl[i+1]==
compOff /\
temporalAcceptConstraint_Traj_last600[i
]==tempActive",
24             "lower_bound": 600,
25             "upper_bound": 600
26         },
27         "first60": {
28             "accepting_event1_constraint": "proj1[0]==projOff",
29             "accepting_event2_constraint": "proj1[i+1]==projOn /\
temporalAcceptConstraint_Traj_first60[i]==
tempActive",
30             "lower_bound": 0,
31             "upper_bound": 60
32         }
33     },
34     "event": {}
35 }
36 }

```

We see that the output is syntactically correct, and the output includes a new temporal constraint labeled **first 60**. We see that the output's accepting event2 constraint is satisfied when the projector turns on. We also see that the lower bound is correctly set to 0 since the user input says *within* 60 seconds.

However, we notice a few errors here where the accepting constraint for the **first60** temporal constraint is `proj1[0]==projOff` when it should indexed at variable i instead of zero as in `proj1[i]==projOff`.

Test prompt 2 Here's the output from the large language model for the second prompt.

```

1 {
2     "initial_state": {
3         "comp": "compOff",
4         "conn": "disconnected",
5         "proj": "projOff"
6     },
7     "end_state": {
8         "maintenance_constraint": "true",
9         "goal_constraint": "compl[i+1]==compOff /\
conn1[i+1]==
disconnected /\
proj1[i+1]==projOff"
10    },
11    "constraints": {
12        "episodes": {

```

```

13     "ProjOn": {
14         "accepting_constraint": "projl[i]==projOn",
15         "state_constraint": "projl[i]==projOn",
16         "lower_bound": 200,
17         "upper_bound": 200
18     },
19     "CompOn": {
20         "accepting_constraint": "compl[i]==compOn",
21         "state_constraint": "compl[i]==compOn",
22         "lower_bound": 300,
23         "upper_bound": 300
24     }
25 },
26 "temporal": {
27     "last600": {
28         "accepting_event1_constraint": "(projl[i]==projOn /\
29         compl[i]==compOn) /\
30         projl[i+1]==projConfirm /\
31         compl[i+1]==compBooting",
32         "accepting_event2_constraint": "projl[i+1]==projOff
33         /\
34         connl[i+1]==disconnected /\
35         compl[i+1]==
36         compOff /\
37         temporalAcceptConstraint_Traj_last600[i]
38         ]==tempActive",
39         "lower_bound": 600,
40         "upper_bound": 600
41     }
42 },
43 "event": {}
44 }
45 }
46 }

```

We observe that the output includes an appropriate new episode constraint labeled **CompOn** for the 300 second episode during which the computer must always be on. Further, we also see that the lower and upper bound for the "ProjOn" constraint are correctly modified to 200.

Chain-of-Thought prompting

Test prompt 1 With CoT prompting, the LLM first gives us an intermediate text output

Initial State

Computer: Off (compOff)

Connection: Disconnected (disconnected)

Projector: Off (projOff)

Episodes

Projector On: The projector must turn on and remain on for 1800 seconds.

Temporal Constraints

Shutdown: Everything (computer, connection, projector) should turn off within 600 seconds after the projector has been on for 1800 seconds.

Projector On Delay: The projector should turn on within 60 seconds from the start.

And then outputs the `state_plan.json`

```
1 {
2   "initial_state": {
3     "comp": "compOff",
4     "conn": "disconnected",
5     "proj": "projOff"
6   },
7   "end_state": {
8     "maintenance_constraint": "true",
9     "goal_constraint": "compl[i+1]==compOff /\ \ connl[i+1]==
10    disconnected /\ \ projl[i+1]==projOff"
11  },
12  "constraints": {
13    "episodes": {
14      "ProjectorOnDuration": {
15        "accepting_constraint": "projl[i]==projOn",
16        "state_constraint": "projl[i]==projOn",
17        "lower_bound": 1800,
18        "upper_bound": 1800
19      }
20    },
21    "temporal": {
22      "ShutdownWithin600": {
23        "accepting_event1_constraint": "projl[i]==projOn /\ \
24        projl[i+1]==projConfirm",
25        "accepting_event2_constraint": "projl[i+1]==projOff
26        /\ \ connl[i+1]==disconnected /\ \ compl[i+1]=compOff
27        /\ \
28        temporalAcceptConstraint_Traj_ShutdownWithin600[i]
29        ]==tempActive",
30        "lower_bound": 600,
31        "upper_bound": 600
32      },
33      "ProjectorOnWithin60": {
34        "accepting_event1_constraint": "projl[i]==projOff /\ \
35        projl[i+1]==projWarmUp",
36        "accepting_event2_constraint": "projl[i+1]==projOn",
37        "lower_bound": 0,
38        "upper_bound": 60
39      }
40    }
41  },
42  "event": {
43  }
44 }
```

```
37 }
38 }
```

We can see that this *state_plan.json* does not make the same mistakes as the direct json generation method and outputs a valid *state_plan.json* that is consistent with the user input provided. The accepting event 1 constraint is correctly indexed with the variable i , and the time at which the projector turns on is constrained by ensuring that the projector state at time $i + 1$ is projector warming up.

We then run this state plan through CTBurton and get the optimal decision variables. The output decision variables can be found in the Appendix.

Test prompt 2 Here's the LLM output for the second prompt, beginning with the intermediate text output

Initial State

Computer: Off (compOff)

Connection: Disconnected (disconnected)

Projector: Off (projOff)

Episodes

Projector On Duration: The projector must turn on and stay on for 200 seconds.

Computer On Duration: The computer should stay on for 300 seconds.

Temporal Constraints

Shutdown: All devices (computer, connection, projector) should turn off within 600 seconds after the projector and computer have completed their required durations.

followed by the *state_plan.json*

```
1 {
2   "initial_state": {
3     "comp": "compOff",
4     "conn": "disconnected",
5     "proj": "projOff"
6   },
7   "end_state": {
8     "maintenance_constraint": "true",
9     "goal_constraint": "compl[i+1]==compOff /\ \ connl[i+1]==
10    disconnected /\ \ projl[i+1]==projOff"
11  },
12  "constraints": {
13    "episodes": {
14      "ProjectorOnDuration": {
15        "accepting_constraint": "projl[i]==projOn",
16        "state_constraint": "projl[i]==projOn",
17        "lower_bound": 200,
18        "upper_bound": 200
19      },
20    }
21  }
22 }
```

```

19     "ComputerOnDuration": {
20         "accepting_constraint": "compl[i]==compOn",
21         "state_constraint": "compl[i]==compOn",
22         "lower_bound": 300,
23         "upper_bound": 300
24     }
25 },
26 "temporal": {
27     "ShutdownWithin600": {
28         "accepting_event1_constraint": "proj1[i]==projOn /\
compl[i]==compOn",
29         "accepting_event2_constraint": "proj1[i+1]==projOff
/\
conn1[i+1]==disconnected /\
compl[i+1]=compOff
/\
temporalAcceptConstraint_Traj_ShutdownWithin600[i]
]==tempActive",
30         "lower_bound": 600,
31         "upper_bound": 600
32     }
33 },
34 "event": {
35
36 }
37 }
38 }

```

which is the expected output and is consistent with the user specified input task.

5.2 Multi-Agent Assembly

Next, we apply our framework to the problem of multi-agent assembly. We look at a simple block stacking problem.

5.2.1 Initial in-context learning prompts

```

1 Natural Language Input -
2
3 Both robot arms start in the home state. Block C is on the floor.
  Block B is stacked on top of block C, and block A is stacked on
  top of block B. Pick up block A and place it on the floor.
4
5 state_plan.json
6
7 {
8     "initial_state":
9     {

```

```

10     "arm1" : "home",
11     "arm2" : "home",
12     "blockAabove" : "nothing",
13     "blockAbelow" : "blockB",
14     "blockBabove" : "blockA",
15     "blockBbelow" : "blockC",
16     "blockCabove" : "blockB",
17     "blockCbelow" : "floor",
18     "floorabove" : "blockC"
19
20 },
21 "end_state":
22 {
23     "maintenance_constraint" : "true",
24     "goal_constraint" : "arm1l[i+1]==home /\ \ arm2l[i+1]==home
        /\ \ blockAabove[i+1]==nothing /\ \ blockAbelow[i+1]==floor
        /\ \ blockBabove[i+1]==nothing /\ \ blockBbelow[i+1]==blockC
        /\ \ blockCabove[i+1]==blockB /\ \ blockCbelow[i+1]==floor
        /\ \ floorabove[i+1]==blockA"
25 },
26
27 }

```

5.2.2 Test prompt

Both robot arms start in the home state. Block C is on the floor. Block B is stacked on top of block C, and block A is stacked on top of block B. Pick up block A and place it on the floor. Place B on top of block A.

```

1 {
2     "initial_state":
3     {
4         "arm1" : "home",
5         "arm2" : "home",
6         "blockAabove" : "nothing",
7         "blockAbelow" : "blockB",
8         "blockBabove" : "blockA",
9         "blockBbelow" : "blockC",
10        "blockCabove" : "blockB",
11        "blockCbelow" : "floor",
12        "floorabove" : "blockC"
13
14    },
15    "end_state":
16    {
17        "maintenance_constraint" : "true",
18        "goal_constraint" : "arm1l[i+1]==home /\ \ arm2l[i+1]==home
        /\ \ blockAabove[i+1]==nothing /\ \ blockAbelow[i+1]==floor

```

```

19     /\blockBabove[i+1]==nothing /\blockBbelow[i+1]==blockC
20     /\blockCabove[i+1]==blockB /\blockCbelow[i+1]==floor
21     /\floorabove[i+1]==blockA"
22 },
23 "constraints":
24 {
25     "episodes":
26     {
27         "Arm1PickBlockA": {
28             "accepting_constraint": "arm1l[i]==home",
29             "state_constraint": "arm1l[i]==pickingA",
30             "lower_bound": 0,
31             "upper_bound": 2500
32         }
33     },
34     "temporal":
35     {
36     },
37     "event":
38     {
39     }
40 }

```

5.3 Search-and-Rescue

Finally, we look at the search and rescue scenario with two different ground agents that must rescue humans from a building.

5.3.1 Initial in-context learning prompts

Most of the prompt is similar to the previous scenarios the same so we exclude repeating the prompt and only include the sections specific to this scenario which includes the natural language input and the state plan JSON file.

```

1 Natural language Input -
2
3 Both the robots are initially outside the building. The first robot
  must rescue person 1 and the second robot must rescue person 2.
  Doors 1 and 2 are initially closed, while Door 3 is already open.
4
5
6 state_plan.json
7

```

```

8 {
9   "initial_state":
10  {
11    "robot1" : "outside",
12    "robot2" : "outside"
13    "door1" : "closed",
14    "door2" : "closed",
15    "door3" : "open"
16  },
17  "end_state":
18  {
19    "maintenance_constraint" : "true",
20    "goal_constraint" : "robot1l[i+1]==rescuedPerson1 /\ \ robot2l
21    [i+1]==rescuedPerson2"
22  },
23 }

```

5.3.2 Test prompts

Test prompt 1 Both the robots are initially outside the building. Either robots can rescue either persons as long as both of them are rescued.

```

1 {
2   "initial_state":
3   {
4     "robot1" : "outside",
5     "robot2" : "outside"
6     "door1" : "closed",
7     "door2" : "closed",
8     "door3" : "open"
9   },
10  "end_state":
11  {
12    "maintenance_constraint" : "true",
13    "goal_constraint" : "(robot1l[i+1]==rescuedPerson1 /\ \
14    robot2l[i+1]==rescuedPerson2) \ / (robot1l[i+1]==
15    rescuedPerson1 /\ \ robot2l[i+1]==rescuedPerson2)"
16  },
17 }

```

In the output we observe that robot 1 rescues person 2 and robot 2 rescues person 1. Person 2 is rescued in 13 seconds whereas Person 1 is rescued in 14 seconds.

Test prompt 2 Both the robots are initially outside the building. The robots must work together to rescue Person 1 as fast as possible.

```

1 {
2   "initial_state":

```

```

3   {
4       "robot1" : "outside",
5       "robot2" : "outside",
6       "door1"  : "closed",
7       "door2"  : "closed",
8       "door3"  : "open"
9   },
10  "end_state":
11  {
12      "maintenance_constraint" : "true",
13      "goal_constraint"       : "robot1l[i+1]==rescuedPerson1"
14  },
15  }

```

In the output, we observe that Person 1 is rescued within 11 seconds as opposed to 14 seconds in the previous cases. We observe interesting behavior where without any explicit requests, robot 2 opens the door for robot 1 so that robot 1 can rescue the person as fast as possible.

5.4 Performance assessment

In this section, we attempt to analyze the scalability of our method as the number of automata is scaled up.

In the initial demonstration of the method, we run the presentation example with one computer, connection, and projector each. In this section, we scale up the number of each type of automata and analyze the amount of time it takes to find a solution to the encoding. Note that the computation times are for the amount of time to find the *first* satisfiable solution. Finding the optimal solution typically takes significantly longer and ctBurton wasn't able to find the solution for problem scales 4x and higher. DNF stands for "Did not finish" which implies that the solver timed out after 2 hours and wasn't able to find a satisfiable solution.

Problem Scale	# Decision Variables	Computation Time (in seconds)
1x	1978	0.404
2x	3236	9.818
4x	5800	79.142
8x	10896	DNF

Table 5.1: Computation Time (in seconds) for various problem scales for the presentation example

Chapter 6

Conclusion and Future Work

In this thesis, we presented a dialogue-driven multi-agent generative activity planning framework that gives us a full state and control trajectory for an action model, parameterized concurrent time automata and a state plan, specifying a sketch for its state evolution. We use the state-of-the-art large language models to aid the state plan generation process from user input in natural language, which is then processed by our generative planner CTBurton.

In this chapter we point out some of the key takeaways from our experimentation, followed by our method’s limitations and directions for future research.

6.1 Takeaways

- Large language models have unlocked a very powerful ability for us to build systems that can interact with users in natural language. Instead of specifying tasks to a system in a domain specific language which is difficult for laypersons to learn, the average consumer can now *code in natural language*. These statistical learning methods are more robust to noisy input, however also do not provide rigorous performance guarantees. While systems like Uhura[6], [7] have been designed in the past, they do not perform generative planning like CTBurton and language models enable more effective dialogue driven systems.
- A promising direction, which we explore in this thesis, is to integrate these statistical learning methods in the form of large language models (that generalize well but don’t always guarantee soundness of output) with generative planners, based on discrete optimization, such CTBurton which are complete and sound over bounded time horizons given the action model and the state plan. As we saw in the related work section, AutoTAMP is a method that uses a similar approach of translating natural language text to an intermediate task representation and then using a PDDL generative planner. However, they use signal temporal logic which is cumbersome to use for the layperson. We use state plans as the task specification format for the generative planner which are more intuitive.
- One of the most promising directions in taking such a *neuro-symbolic* approach has

been to use large language models as *generators* in combination with *verifiers* where the language model generates potential solution candidates, which are fed into a formal verification method that certifies/verifies correctness.

LLMs are particularly useful when planning requires reliance on the 'commonsense knowledge' based on their vast amount of training data. This is helpful when the input from the user does not have sufficient information in order to conclusively deduce the plan, so the LLM can output the most likely plan instead. As a result, these language models are good approximate candidate generators and their outputs can be used as initializations or starting points which can then be refined by solvers or local search. In a way, large language models are extremely large differentiable lookup tables for predicting the next token and are particularly effective for extremely high dimensional search problems where the neural network effectively acts as a heuristic.

- The method presented in this thesis could be interpreted as a variant of this pattern, where we use a large language model to *translate* or *generate* a given user input in natural language to a *state plan* specification. This mapping is probabilistic and susceptible to out-of-distribution inputs. However, the mapping from the state plan to the full optimization encoding is completely deterministic.
- To enhance the robustness of our method, we attempt to minimize the extent to which the mapping is probabilistic (since we cannot guarantee LLM outputs). In our approach, this takes the form of only generating a few lines of MiniZinc code in the state plan as opposed to generating the full encoding from the user input.
- ctBurton is guaranteed to be sound, complete and optimal given the action model and the state plan, of the output state and control trajectory.

6.2 Limitations

Since our method integrates deep learning methods with discrete optimization methods, we inherit both their strengths as well as limitations.

- The translation of the user input in natural language to state plan JSON files relies heavily on the *emergent* in-context learning ability of these large language models which can be quite fragile depending on the number and variety of sample prompts used as context to generate the JSON files for the test prompts. We note, however, that most state-of-the-art models like ChatGPT are exceptionally good at generalizing to novel input based on very few samples.
- Deploying this method in a production/commercial setting will require further iterative engineering development and gradual expansion of the set of input commands mapping to JSON state plans which will make the method significantly more robust. Like most learning methods, increasing the size of a diverse and well curated dataset should help the system asymptotically approach a "mostly perfect, not always correct but still useful" system. We also note that there may be prompting techniques developed in

the future beyond chain-of-thought reasoning that lead to more consistently accurate outputs.

- For complex enough problems, it’s possible that the context window length may not be sufficient. However, this is a very active area of research and a focal point of many large engineering efforts and we anticipate that this will not be bottleneck in the future.
- While having a formal specification of an action model allows us to guarantee that the output is sound, hand-specifying these action models can be quite tedious and time-consuming. Further, it’s possible that the models are too conservative in their expressivity and do not reflect the full set locations or actions that the agents can be in. While the ctBurton plans are optimal for the given state plan or action model, this could lead to plans that are optimal conditioned on the action model, but sub-optimal (or worse, incorrect) in practice.
- The mapping from the state plan JSON to the full minizinc encoding is deterministic, however on some state plan inputs we notice that the GeCode solver can fail with floating point errors despite the state plan being correct. It may be worth exploring solving the encoding using other commercial off-the-shelf solvers such as Gurobi or Mosek.

6.3 Future Work

The limitations of our work and potential applications of the method inspire many different future research directions. Here we summarize some of them.

- **Obtaining action models from large language models**

As highlighted in the limitations section, writing the action models for a given scenario and encoding it into CTBurton can be quite tedious and time-consuming. In this work, we leveraged large language models to translate task specifications from the user in natural language to state plans that can be solved using CTBurton.

Given that, the natural question is to ask whether one could also obtain the action model for a set of concurrent automata from a large language model. This seems a bit more non-trivial compared to the case of a state plan since it’s not clear how one could go about grounding the action model in reality. Perhaps the user provides a description of the agents in words which are then translated to a somewhat structured action representation like PDDL, timeline compatibilities which can then be mapped to parameterized concurrent timed automata.

- **Actively improving action models by interacting with the real world**

The abilities of agents in the real world are often not static and if we aim to build useful general-purpose systems, we must develop a method for improving and expanding the scope of the action model as the agents are exposed to new problems and

scenarios. Concretely, this would imply adding new nodes to the action model's graph representing the idea that the agent is capable of entering new states upon learning new skills which would translate to new edges/transitions in the graph. For instance, an agent might learn new skill primitives such as jumping, or using a hammer or breaking rocks which could be added to the action model and then used to generate more expressive/efficient plans in the future. Another approach could be to analyze the planning domain encoding to identify gaps that need to be closed by searching for new action types.

- **A feedback loop between the action model and the motion planner**

This thesis tackles the higher-level activity planning problem and ignores the lower-level motion planning problem. As a result it's likely that the high level plan generated by CTBurton may not always be executable in the real world. For instance, consider that if the high level action trajectory involves executing a "go-to" command from point A to point B in the environment, however, the path of the agent in the physical world is blocked such that the agent is incapable of executing the command. In this case, the motion planner could report a conflict back to CTBurton which could remove the corresponding edge from the action model and generate a new plan that is feasible. Scotty[38] and Magellan[39] are two examples of methods that tackle this interplay.

- **Action plans in addition to state plans**

While state plans are a more intuitive and interpretable way of specifying tasks as opposed to other methods such as temporal logic, it can sometimes be difficult to explicitly specify states. In addition to a state plan, one could also obtain an approximate *action plan* from the user's input which specifies a rough sequence of the actions that the agent should take to go from the initial state to the final state which could be encoded as a precedence constraint for the full action trajectory. Another closely related direction could be to perform active learning to evaluate the correctness of a model with respect to a chance constraint and refine it. We could also generalize the state plans further to be a unification of action and state plans.

- **Faster (but less expressive) CTBurton encodings**

Another direction is to implement a faster (but slightly less expressive) CTBurton deterministic encoding that represents a temporal opsat encoding which can be solved more efficiently by using specialized subsolvers.

- **Incorporating uncertainty into CTBurton**

Extending the CTBurton encoding implementation to account for uncertainty by assuming that the uncontrollable clock parameters are drawn from a statistical distribution (say gaussian with some mean and variance) or bounded to be within a set.

Chapter 7

Appendix

7.1 Code

The code for this thesis that includes the experimental demonstrations for the 3 examples, the various prompts used with the LLMs, MiniZinc encodings and the json-to-minizinc compiler can be found at <https://git.mers.csail.mit.edu/mers/grads/ctburton>.

7.2 Output decision variables

7.2.1 Presentation System

Initial in-context prompt output decision variables

```
1
2 compl = [0: compOff, 1: compOff, 2: compBooting, 3: compBooting, 4: compOn, 5: compOn, 6: compShutdown, 7: compShutdown,
3         8: compOff, 9: compOff];
4 connl = [0: disconnected, 1: connected, 2: connected, 3: connected, 4: connected, 5: connected, 6: connected, 7:
5         disconnected, 8: disconnected, 9: disconnected];
6 projl = [0: projOff, 1: projWarmUp, 2: projWarmUp, 3: projOn, 4: projOn, 5: projConfirm, 6: projCoolDown, 7: projCoolDown,
7         8: projCoolDown, 9: projOff];
8 compu = [0: compTurnOff, 1: compTurnOn, 2: compTurnOn, 3: compTurnOn, 4: compTurnOn, 5: compTurnOff, 6: compTurnOn, 7:
9         compTurnOn, 8: compTurnOff, 9: compTurnOn];
10 connu = [0: connect, 1: connect, 2: connect, 3: connect, 4: connect, 5: connect, 6: disconnect, 7: disconnect, 8:
11         disconnect, 9: connect];
12 proju = [0: projTurnOn, 1: projTurnOn, 2: projTurnOn, 3: projTurnOn, 4: projTurnOff, 5: projTurnOff, 6: projTurnOn, 7:
13         projTurnOn, 8: projTurnOn, 9: projTurnOn];
14 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: waiting, 7:
15         waiting, 8: waiting, 9: accepted];
16 endStateAcceptAutomatonTrTraj =
17 [
18   0:      1:      2:      3:      4:      5:      6:      7:      8:      9:
19   | 28: true,  true,  true,  true,  true,  true,  true,  true,  false, true
20   | 29: false, false, false, false, false, false, false, false, false, false
21   | 30: false, false, false, false, false, false, false, false, false, false
22   | 31: false, false, false, false, false, false, false, false, false, false
23   | 32: false, false, false, false, false, false, false, false, false, false
24 ];
25 stageWhenAccepted = 9;
26 epAcceptAutomatonTraj = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epActive, 5: epAccepted, 6: epAccepted,
27   , 7: epAccepted, 8: epAccepted, 9: epAccepted];
28 epAcceptAutomatonTrTraj =
29 [
30   0:      1:      2:      3:      4:      5:      6:      7:      8:      9:
31   | 33: true,  true,  true,  false, false, false, false, false, false, true
32   | 34: false, false, false, false, false, true,  true,  true,  true, false
33   | 35: false, false, false, false, false, false, false, false, false, false
34   | 36: false, false, false,  true, false, false, false, false, false, false
35   | 37: false, false, false, false, false, false, false, false, false, false
36   | 38: false, false, false, false,  true, false, false, false, false, false
37   | 39: false, false, false, false, false, false, false, false, false, false
38 ];
39 epAcceptStartEvent = 4;
40 epAcceptEndEvent = 5;
41 epAcceptClockWhenStart = 30.0;
```

```

32 epAcceptClockWhenEnd = 1830.0;
33 epAcceptClockParameter = 1800.0;
34 temporalAcceptConstraint_2_Traj = [0: tempWaiting, 1: tempWaiting, 2: tempWaiting, 3: tempWaiting, 4: tempWaiting, 5:
tempActive, 6: tempActive, 7: tempActive, 8: tempActive, 9: tempAccepted];
35 temporalAcceptConstraint_2_TrTraj =
36 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
37 | 41: true, true, true, true, false, false, false, false, false, true
38 | 42: false, false, false, false, false, false, false, false, false, false
39 | 43: false, false, false, false, true, false, false, false, false, false
40 | 44: false, false, false, false, false, true, true, true, false, false
41 | 45: false, false, false, false, false, false, false, false, true, false
42 | 46: false, false, false, false, false, false, false, false, false, false
43 |];
44 temporalAcceptConstraint_2_event1 = 5;
45 temporalAcceptConstraint_2_event2 = 9;
46 temporalAcceptConstraint_2_event1Clock = 1830.0;
47 temporalAcceptConstraint_2_event2Clock = 2430.0;
48 temporalAcceptConstraint_2_ClockWhenActivated = 1830.0;
49 temporalAcceptConstraint_2_ClockWhenAccepted = 2430.0;
50 temporalAcceptConstraint_2_ClockParameter = 600.0;
51 e = [0: -0.0, 1: 4.940656458412465e-324, 2: 15.0, 3: 30.0, 4: 30.0, 5: 1830.0, 6: 2370.0, 7: 2370.0, 8: 2400.0, 9: 2430.0,
10: 2430.0];
52 c =
53 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
54 | comp: -0.0, 15.0, 15.0, 15.0, 1800.0, 2340.0, -0.0, 30.0, 29.999999999999996, -0.0
55 | conn: -0.0, 15.0, 30.0, 30.0, 1830.0, 2370.0, 2370.0, 29.999999999999996, 59.999999999999996, -0.0
56 | proj: -0.0, 15.0, 30.0, -0.0, 1800.0, 540.0, -0.0, 29.999999999999996, 60.0, -0.0
57 |];
58 co =
59 [ | 0: 1: 2: 3: 4:
60 | comp: -0.0, 5: -0.0, 6: 15.0, 8: 9: 10: 15.0, 30.0,
30.0, 2370.0, 2370.0, 2400.0, 2400.0, -0.0
61 | conn: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4
.940656458412465e-324, 2370.0, 2370.0, -0.0
62 | proj: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 30.0, 30.0,
1830.0, 2370.0, 2370.0, 2430.0, -0.0
63 |];
64 cl =
65 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
66 | comp: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
67 | conn: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
68 | proj: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
69 |];
70 cu =
71 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
72 | comp: -0.0, 15.0, 15.0, 30.0, 1800.0, 2370.0, 2370.0, 2400.0, 2400.0, 29.999999999999996
73 | conn: 4.940656458412465e-324, 15.0, 30.0, 30.0, 1830.0, 2370.0, 2370.0, 2370.0, 2370.0, 59.999999999999996
74 | proj: 4.940656458412465e-324, 15.0, 30.0, 30.0, 1830.0, 2370.0, 2370.0, 2370.0, 2430.0, -0.0
75 |];
76 Eno =
77 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
78 | false, false, false, false, false, false, false, false, false, false
79 | false, false, false, false, false, false, false, false, false, false
80 | false, false, false, false, false, false, false, false, false, false
81 | false, false, false, false, false, false, false, false, false, false
82 | false, false, false, false, true, true, false, false, false, false
83 | false, false, false, false, false, false, true, true, false, false
84 | true, true, false, false, false, false, false, false, true, true
85 | false, false, true, true, false, false, false, false, false, false
86 | false, false, false, false, false, false, false, false, false, false
87 | false, false, false, false, false, false, false, false, false, false
88 | false, true, true, true, true, true, true, true, false, false
89 | true, false, false, false, false, false, false, true, true, true
90 | false, false, false, false, false, false, false, false, false, false
91 | false, false, false, false, false, false, false, false, false, false
92 | false, false, false, false, false, false, false, false, false, false
93 | false, false, false, false, false, false, false, false, false, false
94 | false, false, false, false, false, false, false, false, false, false
95 | false, false, false, false, false, false, false, false, false, false
96 | false, false, false, false, false, false, false, false, false, false
97 | false, false, false, false, false, false, false, false, false, false
98 | true, false, false, false, false, false, false, false, false, true
99 | false, true, true, false, false, false, false, false, false, false
100 | false, false, false, true, true, true, false, false, false, false
101 | false, false, false, false, false, true, false, false, false, false
102 | false, false, false, false, false, false, true, true, true, false
103 | false, false, false, false, false, false, false, false, false, false
104 | false, false, false, false, false, false, false, false, false, false
105 |];
106 Ene =
107 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
108 | false, false, false, false, false, true, false, false, false, true
109 | false, false, false, false, false, false, false, true, false, false
110 | false, true, false, false, false, false, false, false, false, false
111 | false, false, false, true, false, false, false, false, false, false
112 | false, false, false, false, true, false, false, false, false, false
113 | false, false, false, false, false, false, true, false, false, false
114 | true, false, false, false, false, false, false, false, true, false
115 | false, false, true, false, false, false, false, false, false, false
116 | false, false, false, false, false, false, true, false, false, true
117 | true, false, false, false, false, false, false, false, false, false

```

```

118 | false, true, true, true, true, true, false, false, false, false
119 | false, false, false, false, false, false, false, true, true, false
120 | true, false, false, false, false, false, false, false, false, true
121 | false, false, true, false, false, false, false, false, false, false
122 | false, false, false, false, true, false, false, false, false, false
123 | false, false, false, false, false, false, false, false, false, false
124 | false, false, false, false, false, true, false, false, false, false
125 | false, false, false, false, false, false, false, false, false, false
126 | false, false, false, false, false, false, false, false, false, false
127 | false, false, false, false, false, false, false, false, false, false
128 | false, false, false, false, false, false, false, false, false, false
129 | false, true, false, false, false, false, false, false, false, false
130 | false, false, false, true, false, false, false, false, false, false
131 | false, false, false, false, false, false, false, false, false, false
132 | false, false, false, false, false, false, true, true, false, false
133 | false, false, false, false, false, false, false, false, false, false
134 | false, false, false, false, false, false, false, false, true, false
135 |];

```

Test prompt 1 output decision variables

This output is for the state plan using chain-of-thought prompting since the direct JSON generation output was incorrect.

```

1  compl = [0: compOff, 1: compOff, 2: compBooting, 3: compBooting, 4: compOn, 5: compOn, 6: compShutdown, 7: compOff, 8:
    compOff, 9: compOff];
2  conn1 = [0: disconnected, 1: connected, 2: connected, 3: connected, 4: connected, 5: connected, 6: connected, 7: connected
    , 8: disconnected, 9: disconnected];
3  proj1 = [0: projOff, 1: projWarmUp, 2: projWarmUp, 3: projOn, 4: projOn, 5: projConfirm, 6: projCoolDown, 7: projCoolDown,
    8: projOff, 9: projOff];
4  compu = [0: compTurnOn, 1: compTurnOn, 2: compTurnOn, 3: compTurnOn, 4: compTurnOn, 5: compTurnOff, 6: compTurnOn, 7:
    compTurnOff, 8: compTurnOff, 9: compTurnOn];
5  connu = [0: connect, 1: connect, 2: connect, 3: connect, 4: connect, 5: connect, 6: connect, 7: disconnect, 8: disconnect,
    9: connect];
6  proju = [0: projTurnOn, 1: projTurnOn, 2: projTurnOn, 3: projTurnOn, 4: projTurnOff, 5: projTurnOff, 6: projTurnOn, 7:
    projTurnOn, 8: projTurnOff, 9: projTurnOn];
7  endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: waiting, 7:
    waiting, 8: accepted, 9: accepted];
8  endStateAcceptAutomatonTrTraj =
9  [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
10 | 28: true, true, true, true, true, true, true, false, false, true
11 | 29: false, false, false, false, false, false, false, true, false, false
12 | 30: false, false, false, false, false, false, false, false, false, false
13 | 31: false, false, false, false, false, false, false, false, true, false
14 | 32: false, false, false, false, false, false, false, false, false, false
15 |];
16 stageWhenAccepted = 8;
17 epAcceptAutomatonTraj_ProjectorOnDuration = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epActive, 5:
    epAccepted, 6: epAccepted, 7: epAccepted, 8: epAccepted, 9: epAccepted];
18 epAcceptAutomatonTrTraj_ProjectorOnDuration =
19 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
20 | 33: true, true, true, false, false, false, false, false, false, true
21 | 34: false, false, false, false, false, true, true, true, true, false
22 | 35: false, false, false, false, false, false, false, false, false, false
23 | 36: false, false, false, true, false, false, false, false, false, false
24 | 37: false, false, false, false, false, false, false, false, false, false
25 | 38: false, false, false, false, true, false, false, false, false, false
26 | 39: false, false, false, false, false, false, false, false, false, false
27 |];
28 epAcceptStartEvent_ProjectorOnDuration = 4;
29 epAcceptEndEvent_ProjectorOnDuration = 5;
30 epAcceptClockWhenStart_ProjectorOnDuration = 30.0;
31 epAcceptClockWhenEnd_ProjectorOnDuration = 1830.0;
32 epAcceptClockParameter_ProjectorOnDuration = 1800.0;
33 temporalAcceptConstraint_Traj_ShutdownWithin600 = [0: tempWaiting, 1: tempWaiting, 2: tempWaiting, 3: tempWaiting, 4:
    tempWaiting, 5: tempActive, 6: tempActive, 7: tempAccepted, 8: tempAccepted, 9: tempAccepted];
34 temporalAcceptConstraint_TrTraj_ShutdownWithin600 =
35 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
36 | 41: true, true, true, true, false, false, false, false, false, true
37 | 42: false, false, false, false, false, false, false, false, false, false
38 | 43: false, false, false, false, true, false, false, false, false, false
39 | 44: false, false, false, false, false, true, true, false, false, false
40 | 45: false, false, false, false, false, false, false, true, false, false
41 | 46: false, false, false, false, false, false, false, false, true, false
42 |];
43 temporalAcceptConstraint_event1_ShutdownWithin600 = 5;
44 temporalAcceptConstraint_event2_ShutdownWithin600 = 8;
45 temporalAcceptConstraint_event1Clock_ShutdownWithin600 = 1830.0;
46 temporalAcceptConstraint_event2Clock_ShutdownWithin600 = 2430.0;
47 temporalAcceptConstraint_ClockWhenActivated_ShutdownWithin600 = 1830.0;
48 temporalAcceptConstraint_ClockWhenAccepted_ShutdownWithin600 = 2430.0;
49 temporalAcceptConstraint_ClockParameter_ShutdownWithin600 = 600.0;
50 e = [0: -0.0, 1: 4.9406566458412465e-324, 2: 15.0, 3: 30.0, 4: 30.0, 5: 1830.0, 6: 2370.0, 7: 2400.0, 8: 2430.0, 9: 2430.0,
    10: 2430.0];
51 temporalAcceptConstraint_Traj_ProjectorOnWithin60 = [0: tempWaiting, 1: tempActive, 2: tempActive, 3: tempActive, 4:
    tempAccepted, 5: tempAccepted, 6: tempAccepted, 7: tempAccepted, 8: tempAccepted, 9: tempAccepted];

```

```

52 temporalAcceptConstraint_TrTraj_ProjectorOnWithin60 =
53 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
54 | 41: false, false, false, false, false, false, false, false, false, false, true
55 | 42: false, false, false, false, false, false, false, false, false, false, false
56 | 43: true, false, false, false, false, false, false, false, false, false, false
57 | 44: false, true, true, true, false, false, false, false, false, false, false
58 | 45: false, false, false, true, false, false, false, false, false, false, false
59 | 46: false, false, false, false, true, true, true, true, true, true, false
60 ];
61 temporalAcceptConstraint_event1_ProjectorOnWithin60 = 1;
62 temporalAcceptConstraint_event2_ProjectorOnWithin60 = 4;
63 temporalAcceptConstraint_event1Clock_ProjectorOnWithin60 = 4.940656458412465e-324;
64 temporalAcceptConstraint_event2Clock_ProjectorOnWithin60 = 30.0;
65 temporalAcceptConstraint_ClockWhenActivated_ProjectorOnWithin60 = 4.940656458412465e-324;
66 temporalAcceptConstraint_ClockWhenAccepted_ProjectorOnWithin60 = 30.0;
67 temporalAcceptConstraint_ClockParameter_ProjectorOnWithin60 = 30.0;
68 c =
69 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
70 | comp: -0.0, 15.0, 15.0, 15.0, 1800.0, 2340.0, 30.0, 29.999999999999996, 29.999999999999996, -0.0
71 | conn: -0.0, 15.0, 30.0, 30.0, 1830.0, 2370.0, 2400.0, 2430.0, -0.0, -0.0
72 | proj: -0.0, 15.0, 30.0, -0.0, 1800.0, 540.0, 29.999999999999996, 60.0, -0.0, -0.0
73 ];
74 co =
75 [ 0: 1: 2: 3: 4:
76 | comp: -0.0, 5: -0.0, 6: 15.0, 7: 15.0, 8: 2400.0, 9: 2400.0, 10: 2400.0, 11: -0.0
77 | conn: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 2430.0, 2430.0, -0.0
78 | proj: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 2370.0, 2370.0, 2430.0, 2430.0, 2430.0, -0.0
79 ];
80 cl =
81 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
82 | comp: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
83 | conn: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
84 | proj: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
85 ];
86 cu =
87 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
88 | comp: -0.0, 15.0, 15.0, 30.0, 1800.0, 2370.0, 2400.0, 2400.0, 2400.0, 29.999999999999996
89 | conn: 4.940656458412465e-324, 15.0, 30.0, 30.0, 1830.0, 2370.0, 2400.0, 2430.0, 2430.0, -0.0
90 | proj: 4.940656458412465e-324, 15.0, 30.0, 30.0, 1830.0, 2370.0, 2370.0, 2430.0, 2430.0, -0.0
91 ];
92 Eno =
93 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
94 | false, false, false, false, false, false, false, false, false, false, false
95 | false, false, false, false, false, false, false, false, false, false, false
96 | false, false, false, false, false, false, false, false, false, false, false
97 | false, false, false, false, false, false, false, false, false, false, false
98 | false, false, false, false, false, true, true, false, false, false, false
99 | false, false, false, false, false, false, true, false, false, false, false
100 | true, true, false, false, false, false, false, true, true, true, true
101 | false, false, true, true, false, false, false, false, false, false, false
102 | false, false, false, false, false, false, false, false, false, false, false
103 | false, false, false, false, false, false, false, false, false, false, false
104 | false, true, true, true, true, true, true, true, true, false, false
105 | true, false, false, false, false, false, false, false, false, true, true
106 | false, false, false, false, false, false, false, false, false, false, false
107 | false, false, false, false, false, false, false, false, false, false, false
108 | false, false, false, false, false, false, false, false, false, false, false
109 | false, false, false, false, false, false, false, false, false, false, false
110 | false, false, false, false, false, false, false, false, false, false, false
111 | false, false, false, false, false, false, false, false, false, false, false
112 | false, false, false, false, false, false, false, false, false, false, false
113 | false, false, false, false, false, false, false, false, false, false, false
114 | true, false, false, false, false, false, false, false, false, true, true
115 | false, true, true, false, false, false, false, false, false, false, false
116 | false, false, false, true, true, false, false, false, false, false, false
117 | false, false, false, false, false, false, true, false, false, false, false
118 | false, false, false, false, false, false, true, true, false, false, false
119 | false, false, false, false, false, false, false, false, false, false, false
120 | false, false, false, false, false, false, false, false, false, false, false
121 ];
122 Ene =
123 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
124 | false, false, false, false, false, true, false, false, false, true, true
125 | false, false, false, false, false, false, true, false, false, false, false
126 | false, true, false, false, false, false, false, false, false, false, false
127 | false, false, false, true, true, false, false, false, false, false, false
128 | false, false, false, false, true, false, false, false, false, false, false
129 | false, false, false, false, false, false, false, false, false, false, false
130 | true, false, false, false, false, false, false, true, true, false, false
131 | false, false, true, false, false, false, false, false, false, false, false
132 | false, false, false, false, false, false, false, true, false, true, true
133 | true, false, false, false, false, false, false, false, false, false, false
134 | false, true, true, true, true, true, true, true, false, false, false
135 | false, false, false, false, false, false, false, false, true, true, true
136 | true, false, false, false, false, false, false, false, false, false, true
137 | false, false, true, false, false, false, false, false, false, false, false
138 | false, false, false, false, true, false, false, false, false, false, false
139 | false, false, false, false, false, false, false, false, false, false, false

```



```

140 | false, false, false, false, false, true, false, false, false, false
141 | false, false, false, false, false, false, false, false, false, false
142 | false, false, false, false, false, false, false, false, false, false
143 | false, false, false, false, false, false, false, false, false, false
144 | false, false, false, false, false, false, false, false, true, false
145 | false, true, false, false, false, false, false, false, false, false
146 | false, false, false, true, false, false, false, false, false, false
147 | false, false, false, false, false, false, false, false, false, false
148 | false, false, false, false, false, false, true, false, false, false
149 | false, false, false, false, false, false, false, false, false, false
150 | false, false, false, false, false, false, false, true, false, false
151 |];

```

Test prompt 2 output decision variables

```

1 compl = [0: compOff, 1: compOff, 2: compBooting, 3: compBooting, 4: compOn, 5: compOn, 6: compOn, 7: compShutdown, 8:
  compOff, 9: compOff];
2 conn1 = [0: disconnected, 1: connected, 2: connected, 3: connected, 4: connected, 5: connected, 6: connected, 7: connected
  , 8: connected, 9: disconnected];
3 proj1 = [0: projOff, 1: projWarmUp, 2: projWarmUp, 3: projOn, 4: projOn, 5: projConfirm, 6: projCoolDown, 7: projOff, 8:
  projOff, 9: projOff];
4 compu = [0: compTurnOff, 1: compTurnOn, 2: compTurnOn, 3: compTurnOn, 4: compTurnOn, 5: compTurnOn, 6: compTurnOff, 7:
  compTurnOn, 8: compTurnOff, 9: compTurnOn];
5 connu = [0: connect, 1: connect, 2: connect, 3: connect, 4: connect, 5: connect, 6: connect, 7: connect, 8: disconnect, 9:
  connect];
6 proju = [0: projTurnOn, 1: projTurnOn, 2: projTurnOn, 3: projTurnOn, 4: projTurnOff, 5: projTurnOff, 6: projTurnOn, 7:
  projTurnOff, 8: projTurnOff, 9: projTurnOn];
7 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: waiting, 7:
  waiting, 8: waiting, 9: accepted];
8 endStateAcceptAutomatonTrTraj =
9 [| 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
10 | 28: true, true, true, true, true, true, true, true, true, false, true
11 | 29: false, false, false, false, false, false, false, false, false, true, false
12 | 30: false, false, false, false, false, false, false, false, false, false, false
13 | 31: false, false, false, false, false, false, false, false, false, false, false
14 | 32: false, false, false, false, false, false, false, false, false, false, false
15 |];
16 stageWhenAccepted = 9;
17 epAcceptAutomatonTraj_ProjectorOnDuration = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epActive, 5:
  epAccepted, 6: epAccepted, 7: epAccepted, 8: epAccepted, 9: epAccepted];
18 epAcceptAutomatonTrTraj_ProjectorOnDuration =
19 [| 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
20 | 33: true, true, true, false, false, false, false, false, false, true, true
21 | 34: false, false, false, false, false, true, true, true, true, true, false
22 | 35: false, false, false, false, false, false, false, false, false, false, false
23 | 36: false, false, false, true, false, false, false, false, false, false, false
24 | 37: false, false, false, false, false, false, false, false, false, false, false
25 | 38: false, false, false, false, true, false, false, false, false, false, false
26 | 39: false, false, false, false, false, false, false, false, false, false, false
27 |];
28 epAcceptStartEvent_ProjectorOnDuration = 4;
29 epAcceptEndEvent_ProjectorOnDuration = 5;
30 epAcceptClockWhenStart_ProjectorOnDuration = 30.0;
31 epAcceptClockWhenEnd_ProjectorOnDuration = 230.0;
32 epAcceptClockParameter_ProjectorOnDuration = 200.0;
33 epAcceptAutomatonTraj_ComputerOnDuration = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epWaiting, 5:
  epActive, 6: epActive, 7: epAccepted, 8: epAccepted, 9: epAccepted];
34 epAcceptAutomatonTrTraj_ComputerOnDuration =
35 [| 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
36 | 33: true, true, true, true, false, false, false, false, false, true, true
37 | 34: false, false, false, false, false, false, false, true, true, true, false
38 | 35: false, false, false, false, false, false, false, false, false, false, false
39 | 36: false, false, false, false, true, false, false, false, false, false, false
40 | 37: false, false, false, false, false, true, false, false, false, false, false
41 | 38: false, false, false, false, false, false, true, false, false, false, false
42 | 39: false, false, false, false, false, false, false, false, false, false, false
43 |];
44 epAcceptStartEvent_ComputerOnDuration = 5;
45 epAcceptEndEvent_ComputerOnDuration = 7;
46 epAcceptClockWhenStart_ComputerOnDuration = 230.0;
47 epAcceptClockWhenEnd_ComputerOnDuration = 530.0;
48 epAcceptClockParameter_ComputerOnDuration = 300.0;
49 temporalAcceptConstraint_Traj_ShutdownWithin600 = [0: tempWaiting, 1: tempWaiting, 2: tempWaiting, 3: tempWaiting, 4:
  tempWaiting, 5: tempActive, 6: tempActive, 7: tempActive, 8: tempActive, 9: tempAccepted];
50 temporalAcceptConstraint_TrTraj_ShutdownWithin600 =
51 [| 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
52 | 41: true, true, true, true, false, false, false, false, false, true, true
53 | 42: false, false, false, false, false, false, false, false, false, false, false
54 | 43: false, false, false, false, true, false, false, false, false, false, false
55 | 44: false, false, false, false, false, true, true, true, true, false, false
56 | 45: false, false, false, false, false, false, false, false, false, true, false
57 | 46: false, false, false, false, false, false, false, false, false, false, false
58 |];
59 temporalAcceptConstraint_event1_ShutdownWithin600 = 5;
60 temporalAcceptConstraint_event2_ShutdownWithin600 = 9;
61 temporalAcceptConstraint_event1Clock_ShutdownWithin600 = 230.0;
62 temporalAcceptConstraint_event2Clock_ShutdownWithin600 = 830.0;

```

```

63 temporalAcceptConstraint_ClockWhenActivated_ShutdownWithin600 = 230.0;
64 temporalAcceptConstraint_ClockWhenAccepted_ShutdownWithin600 = 830.0;
65 temporalAcceptConstraint_ClockParameter_ShutdownWithin600 = 600.0;
66 e = [0: -0.0, 1: 4.940656458412465e-324, 2: 15.0, 3: 30.0, 4: 30.0, 5: 230.0, 6: 470.0, 7: 530.0, 8: 560.0, 9: 830.0, 10:
      830.0];
67 c =
68 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
69 | comp: -0.0, 15.0, 15.0, 15.0, 200.0, 440.0, 500.0, 30.0, 270.0, -0.0
70 | conn: -0.0, 15.0, 30.0, 30.0, 230.0, 470.0, 530.0, 560.0, 830.0, -0.0
71 | proj: -0.0, 15.0, 30.0, -0.0, 200.0, 240.0, 60.0, 29.999999999999999, 300.0, -0.0
72 ];
73 co =
74 [ 0: 1: 2: 3: 4:
75 | comp: -0.0, 5: -0.0, 6: 15.0, 7: 15.0, 8: 9: 10:
      30.0,
76 | conn: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 4
      .940656458412465e-324, 4.940656458412465e-324, 4.940656458412465e-324, 830.0, -0.0
77 | proj: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 230.0, 470.0, 530.0, 560.0, 830.0, 30.0, -0.0
78 ];
79 ci =
80 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
81 | comp: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
82 | conn: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
83 | proj: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
84 ];
85 cu =
86 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
87 | comp: -0.0, 15.0, 15.0, 30.0, 200.0, 440.0, 530.0, 560.0, 560.0, 270.0
88 | conn: 4.940656458412465e-324, 15.0, 30.0, 30.0, 230.0, 470.0, 530.0, 560.0, 830.0, -0.0
89 | proj: 4.940656458412465e-324, 15.0, 30.0, 30.0, 230.0, 470.0, 530.0, 530.0, 530.0, 300.0
90 ];
91 Eno =
92 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
93 | false, false, false, false, false, false, false, false, false, false
94 | false, false, false, false, false, false, false, false, false, false
95 | false, false, false, false, false, false, false, false, false, false
96 | false, false, false, false, false, false, false, false, false, false
97 | false, false, false, false, true, true, true, false, false, false
98 | false, false, false, false, false, false, false, true, false, false
99 | true, true, false, false, false, false, false, false, true, true
100 | false, false, true, true, false, false, false, false, false, false
101 | false, false, false, false, false, false, false, false, false, false
102 | false, false, false, false, false, false, false, false, false, false
103 | false, true, true, true, true, true, true, true, true, false
104 | true, false, false, false, false, false, false, false, false, true
105 | false, false, false, false, false, false, false, false, false, false
106 | false, false, false, false, false, false, false, false, false, false
107 | false, false, false, false, false, false, false, false, false, false
108 | false, false, false, false, false, false, false, false, false, false
109 | false, false, false, false, false, false, false, false, false, false
110 | false, false, false, false, false, false, false, false, false, false
111 | false, false, false, false, false, false, false, false, false, false
112 | false, false, false, false, false, false, false, false, false, false
113 | true, false, false, false, false, false, false, false, true, true
114 | false, true, true, false, false, false, false, false, false, false
115 | false, false, false, true, true, false, false, false, false, false
116 | false, false, false, false, false, true, false, false, false, false
117 | false, false, false, false, false, false, true, false, false, false
118 | false, false, false, false, false, false, false, false, false, false
119 | false, false, false, false, false, false, false, false, false, false
120 ];
121 Ene =
122 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:
123 | false, false, false, false, false, false, true, false, false, true
124 | false, false, false, false, false, false, false, true, false, false
125 | false, true, false, false, false, false, false, false, false, false
126 | false, false, false, true, false, false, false, false, false, false
127 | false, false, false, false, true, true, false, false, false, false
128 | false, false, false, false, false, false, false, false, false, false
129 | true, false, false, false, false, false, false, false, true, false
130 | false, false, true, false, false, false, false, false, false, false
131 | false, false, false, false, false, false, false, false, true, true
132 | true, false, false, false, false, false, false, false, false, false
133 | false, true, true, true, true, true, true, true, false, false
134 | false, false, false, false, false, false, false, false, false, false
135 | true, false, false, false, false, false, false, false, false, true
136 | false, false, true, false, false, false, false, false, false, false
137 | false, false, false, false, true, false, false, false, false, false
138 | false, false, false, false, false, false, false, false, false, false
139 | false, false, false, false, false, true, false, false, false, false
140 | false, false, false, false, false, false, false, false, false, false
141 | false, false, false, false, false, false, false, false, false, false
142 | false, false, false, false, false, false, false, false, false, false
143 | false, false, false, false, false, false, false, true, true, false
144 | false, true, false, false, false, false, false, false, false, false
145 | false, false, false, true, false, false, false, false, false, false
146 | false, false, false, false, false, false, false, false, false, false
147 | false, false, false, false, false, false, false, false, false, false
148 | false, false, false, false, false, false, false, false, false, false
149 | false, false, false, false, false, true, false, false, false, false

```

7.2.2 Multi-agent Assembly

Initial in-context learning prompt

```

1 arm1 = [0: home, 1: home, 2: pickingA, 3: pickingA, 4: pickedA, 5: placingAonFloor, 6: placedAonFloor];
2 arm1u = [0: doNothing, 1: pickA, 2: doNothing, 3: doNothing, 4: placeAonFloor, 5: doNothing, 6: doNothing];
3 blockAbove = [0: nothing, 1: nothing, 2: nothing, 3: nothing, 4: gripper, 5: nothing, 6: nothing];
4 blockBelow = [0: blockB, 1: nothing, 2: nothing, 3: nothing, 4: gripper, 5: floor, 6: floor];
5 blockBabove = [0: blockA, 1: nothing, 2: nothing, 3: nothing, 4: nothing, 5: nothing, 6: nothing];
6 blockBbelow = [0: blockC, 1: blockC, 2: nothing, 3: nothing, 4: nothing, 5: nothing, 6: blockC];
7 blockCabove = [0: blockB, 1: nothing, 2: nothing, 3: nothing, 4: nothing, 5: nothing, 6: blockB];
8 blockCbelow = [0: floor, 1: floor, 2: floor, 3: floor, 4: floor, 5: floor, 6: floor];
9 floorabove = [0: blockC, 1: nothing, 2: nothing, 3: nothing, 4: nothing, 5: gripper, 6: blockA];
10 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: accepted];
11 endStateAcceptAutomatonTrTraj =
12 []
13 | 28: true, true, true, true, true, false, true
14 | 29: false, false, false, false, false, true, false
15 | 30: false, false, false, false, false, false, false
16 | 31: false, false, false, false, false, false, false
17 | 32: false, false, false, false, false, false, false
18 [];
19 stageWhenAccepted = 6;
20 e = [0: -0.0, 1: 4.940656458412465e-324, 2: 9.881312916824931e-324, 3: 1.48219693752374e-323, 4: 1.0, 5: 1.0, 6: 3.0, 7: 3
    .0];
21 c =
22 []
23 | arm1: -0.0, 4.940656458412465e-324, -0.0, 1.0, -0.0, 2.0, -0.0
24 | arm2: -0.0, -0.0, 4.940656458412465e-324, 1.0, 1.0, 3.0, -0.0
25 [];
26 co =
27 []
28 | arm1: -0.0, 5: -0.0, 9.881312916824931e-324, 9.881312916824931e-324, 1.0,
    1.0, -0.0
29 | arm2: -0.0, 4.940656458412465e-324, 4.940656458412465e-324, 1.48219693752374e-323, 1.48219693752374e-323, 1
    .48219693752374e-323, 1.48219693752374e-323, -0.0
30 [];
31 ci =
32 []
33 | arm1: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
34 | arm2: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
35 [];
36 cu =
37 []
38 | arm1: -0.0, 9.881312916824931e-324, 9.881312916824931e-324, 1.0, 1.0, 3.0, -0.0
39 | arm2: 4.940656458412465e-324, -0.0, 1.48219693752374e-323, 1.0, 1.0, 3.0, 3.0
40 [];
41 Enoi =
42 []
43 | 101: false, false, false, false, false, false, false
44 | 102: true, true, true, true, true, true, true
45 | 103: false, false, false, false, false, false, false
46 | 104: false, false, false, false, false, false, false
47 | 105: false, false, false, false, false, false, false
48 | 106: false, false, false, false, false, false, false
49 | 107: false, false, false, false, false, false, false
50 | 108: false, false, false, false, false, false, false
51 | 109: false, false, false, false, false, false, false
52 | 110: false, false, false, false, false, false, false
53 | 111: false, false, false, false, false, false, false
54 | 112: false, false, false, false, false, false, false
55 | 113: false, false, false, false, false, false, false
56 | 114: false, false, false, false, false, false, false
57 | 115: false, false, false, false, false, false, false
58 | 116: false, false, false, false, false, false, false
59 | 117: false, false, false, false, false, false, false
60 | 118: false, false, false, false, false, false, false
61 | 119: false, false, false, false, false, false, false
62 | 120: false, false, false, false, false, false, false
63 | 121: false, false, false, false, false, false, false
64 | 122: false, false, false, false, false, false, false
65 | 123: false, false, false, false, false, false, false
66 | 124: false, false, false, false, false, false, false
67 | 125: false, false, false, false, false, false, false
68 | 126: false, false, false, false, false, false, false
69 | 127: false, false, false, false, false, false, false
70 | 128: false, false, false, false, false, false, false
71 | 129: false, false, false, false, false, false, false
72 | 130: false, false, false, false, false, false, false
73 | 131: false, false, false, false, false, false, false
74 | 132: false, false, false, false, false, false, false
75 | 133: false, false, false, false, false, false, false

```



```

168 | 124: false, false, false, false, false, false, false
169 | 125: false, false, false, false, false, false, false
170 | 126: false, false, false, false, false, false, false
171 | 127: false, false, false, false, false, false, false
172 | 128: false, false, false, false, false, false, false
173 | 129: false, false, false, false, false, false, false
174 | 130: false, false, false, false, false, false, false
175 | 131: false, false, false, false, false, false, false
176 | 132: false, false, false, false, false, false, false
177 | 133: false, false, false, false, false, false, false
178 | 134: false, false, false, false, false, false, false
179 | 135: false, false, false, false, false, false, false
180 | 136: false, false, false, false, false, false, false
181 | 137: false, false, false, false, false, false, false
182 | 138: false, false, false, false, false, false, false
183 | 139: false, false, false, false, false, false, false
184 | 140: false, false, false, false, false, false, false
185 | 141: false, false, false, false, false, false, false
186 | 142: false, false, false, false, false, false, false
187 | 143: false, false, false, false, false, false, false
188 | 144: false, false, false, false, false, false, false
189 | 145: false, false, false, false, false, false, false
190 | 146: false, false, false, false, false, false, false
191 | 147: false, false, false, false, false, false, false
192 | 148: true, false, false, false, false, false, false
193 | ];
194 Ene2 =
195 [ | 0: 1: 2: 3: 4: 5: 6:
196 | 101: true, false, false, false, false, false, true
197 | 102: false, true, false, true, true, true, false
198 | 103: false, false, false, false, false, false, false
199 | 104: false, false, false, false, false, false, false
200 | 105: false, false, false, false, false, false, false
201 | 106: false, false, false, false, false, false, false
202 | 107: false, false, false, false, false, false, false
203 | 108: false, false, false, false, false, false, false
204 | 109: false, false, false, false, false, false, false
205 | 110: false, false, false, false, false, false, false
206 | 111: false, false, false, false, false, false, false
207 | 112: false, false, false, false, false, false, false
208 | 113: false, false, false, false, false, false, false
209 | 114: false, false, false, false, false, false, false
210 | 115: false, false, false, false, false, false, false
211 | 116: false, false, true, false, false, false, false
212 | 117: false, false, false, false, false, false, false
213 | 118: false, false, false, false, false, false, false
214 | 119: false, false, false, false, false, false, false
215 | 120: false, false, false, false, false, false, false
216 | 121: false, false, false, false, false, false, false
217 | 122: false, false, false, false, false, false, false
218 | 123: false, false, false, false, false, false, false
219 | 124: false, false, false, false, false, false, false
220 | 125: false, false, false, false, false, false, false
221 | 126: false, false, false, false, false, false, false
222 | 127: false, false, false, false, false, false, false
223 | 128: false, false, false, false, false, false, false
224 | 129: false, false, false, false, false, false, false
225 | 130: false, false, false, false, false, false, false
226 | 131: false, false, false, false, false, false, false
227 | 132: false, false, false, false, false, false, false
228 | 133: false, false, false, false, false, false, false
229 | 134: false, false, false, false, false, false, false
230 | 135: false, false, false, false, false, false, false
231 | 136: false, false, false, false, false, false, false
232 | 137: false, false, false, false, false, false, false
233 | 138: false, false, false, false, false, false, false
234 | 139: false, false, false, false, false, false, false
235 | 140: false, false, false, false, false, false, false
236 | 141: false, false, false, false, false, false, false
237 | 142: false, false, false, false, false, false, false
238 | 143: false, false, false, false, false, false, false
239 | 144: false, false, false, false, false, false, false
240 | 145: false, false, false, false, false, false, false
241 | 146: false, false, false, false, false, false, false
242 | 147: false, false, false, false, false, false, false
243 | 148: false, false, false, false, false, false, false
244 | ];
245 -----

```

Test prompt 1

```

1 arm1 = [0: home, 1: pickingA, 2: pickedA, 3: placingAonFloor, 4: placedAonFloor, 5: goingHome, 6: home];
2 arm2 = [0: home, 1: pickingB, 2: home, 3: home, 4: pickingA, 5: pickingB, 6: placedBonA];
3 arm1u = [0: pickA, 1: doNothing, 2: placeAonFloor, 3: doNothing, 4: doNothing, 5: placeAonFloor, 6: doNothing];
4 arm2u = [0: pickB, 1: doNothing, 2: doNothing, 3: pickA, 4: pickB, 5: doNothing, 6: doNothing];
5 blockAbove = [0: nothing, 1: floor, 2: gripper, 3: nothing, 4: nothing, 5: nothing, 6: blockB];
6 blockBelow = [0: blockB, 1: gripper, 2: gripper, 3: gripper, 4: floor, 5: floor, 6: floor];

```

```

7 blockAbove = [0: blockA, 1: nothing, 2: nothing, 3: nothing, 4: nothing, 5: nothing, 6: nothing];
8 blockBelow = [0: blockC, 1: nothing, 2: nothing, 3: nothing, 4: nothing, 5: blockA, 6: blockA];
9 blockAbove = [0: blockB, 1: nothing, 2: nothing, 3: nothing, 4: nothing, 5: nothing, 6: blockB];
10 blockBelow = [0: floor, 1: floor, 2: floor, 3: floor, 4: floor, 5: floor, 6: floor];
11 floorAbove = [0: blockC, 1: nothing, 2: nothing, 3: gripper, 4: blockA, 5: nothing, 6: blockA];
12 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: accepted];
13 endStateAcceptAutomatonTrTraj =
14 | 0: 1: 2: 3: 4: 5: 6:
15 | 28: true, true, true, true, true, false, true
16 | 29: false, false, false, false, false, true, false
17 | 30: false, false, false, false, false, false, false
18 | 31: false, false, false, false, false, false, false
19 | 32: false, false, false, false, false, false, false
20 |];
21 stageWhenAccepted = 6;
22 epAcceptAutomatonTraj = [0: epWaiting, 1: epActive, 2: epAccepted, 3: epAccepted, 4: epAccepted, 5: epAccepted, 6:
epAccepted];
23 epAcceptAutomatonTrTraj =
24 | 0: 1: 2: 3: 4: 5: 6:
25 | 33: false, false, false, false, false, false, true
26 | 34: false, false, true, true, true, true, false
27 | 35: false, false, false, false, false, false, false
28 | 36: true, false, false, false, false, false, false
29 | 37: false, false, false, false, false, false, false
30 | 38: false, true, false, false, false, false, false
31 | 39: false, false, false, false, false, false, false
32 |];
33 epAcceptStartEvent = 1;
34 epAcceptEndEvent = 2;
35 epAcceptClockWhenStart = 4.940656458412465e-324;
36 epAcceptClockWhenEnd = 1.0;
37 epAcceptClockParameter = 1.0;
38 e = [0: -0.0, 1: 4.940656458412465e-324, 2: 1.0, 3: 1.0, 4: 3.0, 5: 3.0, 6: 8.0, 7: 8.0];
39 c =
40 | 0: 1: 2: 3: 4: 5: 6:
41 | arm1: -0.0, 1.0, -0.0, 2.0, -0.0, 5.0, -0.0
42 | arm2: -0.0, 1.0, 1.0, 3.0, 3.0, 8.0, -0.0
43 |];
44 co =
45 | 0: 1: 2: 3: 4: 5: 6: 7:
46 | arm1: -0.0, 4.940656458412465e-324, 1.0, 1.0, 3.0, 3.0, 8.0, -0.0
47 | arm2: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
48 |];
49 cl =
50 | 0: 1: 2: 3: 4: 5: 6:
51 | arm1: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
52 | arm2: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
53 |];
54 cu =
55 | 0: 1: 2: 3: 4: 5: 6:
56 | arm1: 4.940656458412465e-324, 1.0, 1.0, 3.0, 3.0, 8.0, -0.0
57 | arm2: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
58 |];
59 Enol =
60 | 0: 1: 2: 3: 4: 5: 6:
61 | 101: false, false, false, false, false, false, false
62 | 102: true, true, true, true, true, true, true
63 | 103: false, false, false, false, false, false, false
64 | 104: false, false, false, false, false, false, false
65 | 105: false, false, false, false, false, false, false
66 | 106: false, false, false, false, false, false, false
67 | 107: false, false, false, false, false, false, false
68 | 108: false, false, false, false, false, false, false
69 | 109: false, false, false, false, false, false, false
70 | 110: false, false, false, false, false, false, false
71 | 111: false, false, false, false, false, false, false
72 | 112: false, false, false, false, false, false, false
73 | 113: false, false, false, false, false, false, false
74 | 114: false, false, false, false, false, false, false
75 | 115: false, false, false, false, false, false, false
76 | 116: false, false, false, false, false, false, false
77 | 117: false, false, false, false, false, false, false
78 | 118: false, false, false, false, false, false, false
79 | 119: false, false, false, false, false, false, false
80 | 120: false, false, false, false, false, false, false
81 | 121: false, false, false, false, false, false, false
82 | 122: false, false, false, false, false, false, false
83 | 123: false, false, false, false, false, false, false
84 | 124: false, false, false, false, false, false, false
85 | 125: false, false, false, false, false, false, false
86 | 126: false, false, false, false, false, false, false
87 | 127: false, false, false, false, false, false, false
88 | 128: false, false, false, false, false, false, false
89 | 129: false, false, false, false, false, false, false
90 | 130: false, false, false, false, false, false, false
91 | 131: false, false, false, false, false, false, false
92 | 132: false, false, false, false, false, false, false
93 | 133: false, false, false, false, false, false, false
94 | 134: false, false, false, false, false, false, false
95 | 135: false, false, false, false, false, false, false
96 | 136: false, false, false, false, false, false, false
97 | 137: false, false, false, false, false, false, false

```


7.2.3 Search-and-Rescue

Initial in-context learning prompt

```
1 robot1l = [0: outside, 1: goingToDoor1, 2: atDoor1, 3: door1Opening, 4: insideRoom1, 5: goingToDoor3, 6: goingToDoor3, 7:
  goingToDoor3, 8: atDoor3, 9: atDoor3, 10: goingDoor3Person1, 11: rescuedPerson1, 12: rescuedPerson1, 13:
  rescuedPerson1, 14: rescuedPerson1];
2 robot2l = [0: outside, 1: outside, 2: goingToDoor1, 3: atDoor1, 4: atDoor1, 5: atDoor1, 6: insideRoom1, 7: goingToDoor3,
  8: atDoor3, 9: goingDoor3Person2, 10: goingDoor3Person2, 11: rescuedPerson2, 12: rescuedPerson2, 13: rescuedPerson2,
  14: rescuedPerson2];
3 door1l = [0: closed, 1: closed, 2: closed, 3: opening, 4: open, 5: open, 6: open, 7: open, 8: open, 9: open, 10: open, 11:
  open, 12: open, 13: open, 14: open];
4 door2l = [0: closed, 1: closed, 2: closed, 3: closed, 4: closed, 5: closed, 6: closed, 7: closed, 8: closed, 9: closed,
  10: closed, 11: closed, 12: closed, 13: closed, 14: closed];
5 door3l = [0: open, 1: open, 2: open, 3: open, 4: open, 5: open, 6: open, 7: open, 8: open, 9: open, 10: open, 11: open,
  12: open, 13: open, 14: open];
6 robot1u = [0: goToDoor1, 1: doNothing, 2: openDoor1, 3: doNothing, 4: goToDoor3, 5: doNothing, 6: doNothing, 7: doNothing,
  8: doNothing, 9: goToDoor3Person1, 10: doNothing, 11: doNothing, 12: doNothing, 13: doNothing, 14: doNothing];
7 robot2u = [0: doNothing, 1: goToDoor1, 2: doNothing, 3: doNothing, 4: doNothing, 5: doNothing, 6: goToDoor3, 7: doNothing,
  8: goToDoor3Person2, 9: doNothing, 10: doNothing, 11: doNothing, 12: doNothing, 13: doNothing, 14: doNothing];
8 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: waiting, 7:
  waiting, 8: waiting, 9: waiting, 10: waiting, 11: accepted, 12: accepted, 13: accepted, 14: accepted];
9 endStateAcceptAutomatonTrTraj =
10 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
11 | 28: true, true, true, true, true, true, true, true, true, true, true, false, false, false, false, true
12 | 29: false, false, false, false, false, false, false, false, false, false, false, true, false, false, false, false
13 | 30: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
14 | 31: false, false, false, false, false, false, false, false, false, false, false, false, true, true, true, false
15 | 32: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
16 ];
17 stageWhenAccepted = 11;
18 clockWhenPerson1Rescued = -0.0;
19 clockWhenPerson2Rescued = -0.0;
20 epAcceptAutomatonTraj = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epWaiting, 5: epWaiting, 6: epWaiting,
  7: epWaiting, 8: epWaiting, 9: epWaiting, 10: epWaiting, 11: epWaiting, 12: epWaiting, 13: epWaiting, 14: epWaiting
  ];
21 epAcceptAutomatonTrTraj =
22 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
23 | 33: true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true
24 | 34: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
25 | 35: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
26 | 36: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
27 | 37: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
28 | 38: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
29 | 39: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
30 ];
31 epAcceptStartEvent = 0;
32 epAcceptEndEvent = 0;
33 epAcceptClockWhenStart = -0.0;
34 epAcceptClockWhenEnd = 1.0;
35 epAcceptClockParameter = 1.0;
36 e = [0: -0.0, 1: 4.940656458412465e-324, 2: 1.0, 3: 2.0, 4: 4.0, 5: 4.0, 6: 4.0, 7: 4.0, 8: 8.0, 9: 8.0, 10: 9.0, 11: 14.0
  , 12: 14.0, 13: 14.0, 14: 14.0, 15: 14.0];
37 c =
38 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
39 | robot1: -0.0, 1.0, 1.0, 2.0, -0.0, -0.0, -0.0, 4.0, -0.0, 0.9999999999999999, 5.0, -0.0, -0.0, -0.0, -0.0
40 | robot2: -0.0, 1.0, 1.0, 2.0, 2.0, 2.0, -0.0, 4.0, -0.0, 0.9999999999999999, 6.0, -0.0, -0.0, -0.0, -0.0
41 ];
42 co =
43 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14: 15:
44 | robot1: -0.0, 4.940656458412465e-324, 1.0, 2.0, 4.0, 4.0, 4.0, 4.0, 8.0, 8.0, 9.0, 14.0, 14.0, 14.0, 14.0, -0.0
45 | robot2: -0.0, -0.0, 1.0, 2.0, 2.0, 2.0, 4.0, 4.0, 8.0, 8.0, 8.0, 14.0, 14.0, 14.0, 14.0, -0.0
46 ];
47 cl =
48 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
49 | robot1: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
50 | robot2: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
51 ];
52 cu =
53 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
54 | robot1: 4.940656458412465e-324, 1.0, 2.0, 4.0, 4.0, 4.0, 4.0, 8.0, 8.0, 9.0, 14.0, 14.0, 14.0, 14.0, -0.0
55 | robot2: -0.0, 1.0, 2.0, 2.0, 2.0, 4.0, 4.0, 8.0, 8.0, 8.0, 14.0, 14.0, 14.0, 14.0, -0.0
56 ];
57 Enol =
58 [ 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
59 | 101: true, false, false, false, false, false, false, false, false, false, false, false, false, false, false
60 | 102: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
61 | 103: false, true, false, false, false, false, false, false, false, false, false, false, false, false, false
62 | 104: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
63 | 105: false, false, true, false, false, false, false, false, false, false, false, false, false, false, false
64 | 106: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
65 | 107: false, false, false, true, false, false, false, false, false, false, false, false, false, false, false
66 | 108: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
67 | 109: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
68 | 110: false, false, false, false, true, false, false, false, false, false, false, false, false, false, false
69 | 111: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
70 | 112: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
71 | 113: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
72 | 114: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
```



```

165 | 127: false, false, false, false, false, false, false, true, false, false, false, false, false, false, false
166 | 128: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
167 | 129: false, false, false, false, false, false, false, false, false, true, false, false, false, false, false
168 | 130: false, false, false, false, false, false, true, true, false, false, false, false, false, false, false
169 | 131: false, false, false, false, false, false, false, false, false, false, true, false, false, false, false
170 | 132: false, false, false, false, false, false, false, false, true, false, false, false, false, false, false
171 | 133: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
172 | 134: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
173 | 135: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
174 | 136: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
175 | 137: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
176 | ];
177 Ene2 =
178 [ |
179 | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
180 | 101: true, false, false, false, false, false, false, false, false, false, false, false, false, false, true
181 | 102: false, true, false, false, false, false, false, false, false, false, false, false, false, false, false
182 | 103: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
183 | 104: false, false, true, false, false, false, false, false, false, false, false, false, false, false, false
184 | 105: false, false, false, true, true, false, false, false, false, false, false, false, false, false, false
185 | 106: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
186 | 107: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
187 | 108: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
188 | 109: false, false, false, false, false, true, false, false, false, false, false, false, false, false, false
189 | 110: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
190 | 111: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
191 | 112: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
192 | 113: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
193 | 114: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
194 | 115: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
195 | 116: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
196 | 117: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
197 | 118: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
198 | 119: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
199 | 120: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
200 | 121: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
201 | 122: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
202 | 123: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
203 | 124: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
204 | 125: false, false, false, false, false, false, true, false, false, false, false, false, false, false, false
205 | 126: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
206 | 127: false, false, false, false, false, false, false, true, false, false, false, false, false, false, false
207 | 128: false, false, false, false, false, false, false, false, false, false, false, true, true, true, true
208 | 129: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
209 | 130: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
210 | 131: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
211 | 132: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
212 | 133: false, false, false, false, false, false, false, false, true, false, false, false, false, false, false
213 | 134: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
214 | 135: false, false, false, false, false, false, false, false, false, true, false, false, false, false, false
215 | 136: false, false, false, false, false, false, false, false, false, true, false, false, false, false, false
216 | 137: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
| ];

```

Test prompt 1

```

1 robot1 = [0: outside, 1: goingToDoor1, 2: atDoor1, 3: door1Opening, 4: insideRoom1, 5: goingToDoor3, 6: goingToDoor3,
7: goingToDoor3, 8: goingToDoor3, 9: goingToDoor3, 10: atDoor3, 11: goingDoor3Person2, 12: goingDoor3Person2,
13: rescuedPerson2, 14: rescuedPerson2];
2 robot2 = [0: outside, 1: goingToDoor1, 2: atDoor1, 3: atDoor1, 4: atDoor1, 5: atDoor1, 6: insideRoom1, 7: goingToDoor3,
8: atDoor3, 9: goingDoor3Person1, 10: goingDoor3Person1, 11: goingDoor3Person1, 12: rescuedPerson1, 13:
rescuedPerson1, 14: rescuedPerson1];
3 door1 = [0: closed, 1: closed, 2: closed, 3: opening, 4: open, 5: open, 6: open, 7: open, 8: open, 9: open, 10: open, 11:
open, 12: open, 13: open, 14: open];
4 door2 = [0: closed, 1: closed, 2: closed, 3: closed, 4: closed, 5: closed, 6: closed, 7: closed, 8: closed, 9: closed,
10: closed, 11: closed, 12: closed, 13: closed, 14: closed];
5 door3 = [0: open, 1: open, 2: open, 3: open, 4: open, 5: open, 6: open, 7: open, 8: open, 9: open, 10: open, 11: open,
12: open, 13: open, 14: open];
6 robot1u = [0: goToDoor1, 1: doNothing, 2: openDoor1, 3: doNothing, 4: goToDoor3, 5: doNothing, 6: doNothing, 7: doNothing,
8: doNothing, 9: doNothing, 10: goToDoor3Person2, 11: doNothing, 12: doNothing, 13: doNothing, 14: doNothing];
7 robot2u = [0: goToDoor1, 1: doNothing, 2: doNothing, 3: doNothing, 4: doNothing, 5: doNothing, 6: goToDoor3, 7: doNothing,
8: goToDoor3Person1, 9: doNothing, 10: doNothing, 11: doNothing, 12: doNothing, 13: doNothing, 14: doNothing];
8 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 7:
waiting, 8: waiting, 9: waiting, 10: waiting, 11: waiting, 12: waiting, 13: accepted, 14: accepted];
9 endStateAcceptAutomatonTrTraj =
10 [ |
11 | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
12 | 28: true, true, true, true, true, true, true, true, true, true, true, true, false, false, true
13 | 29: false, false, false, false, false, false, false, false, false, false, false, false, true, false, false
14 | 30: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
15 | 31: false, false, false, false, false, false, false, false, false, false, false, false, false, true, false
16 | 32: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
| ];
17 stageWhenAccepted = 13;
18 clockWhenPerson1Rescued = -0.0;
19 clockWhenPerson2Rescued = -0.0;
20 epAcceptAutomatonTraj = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epWaiting, 5: epWaiting, 6: epWaiting,
7: epWaiting, 8: epWaiting, 9: epWaiting, 10: epWaiting, 11: epWaiting, 12: epWaiting, 13: epWaiting, 14: epWaiting
];

```



```

204 | 126: false, false, false, false, false, false, false, false, false, false, false, false, false, true, true, false
205 | 127: false, false, false, false, false, false, false, false, true, false, false, false, false, false, false, false
206 | 128: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
207 | 129: false, false, false, false, false, false, false, false, true, false, false, false, false, false, false, false
208 | 130: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
209 | 131: false, false, false, false, false, false, false, false, false, false, false, false, true, false, false, false
210 | 132: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
211 | 133: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
212 | 134: false, false, false, false, false, false, false, false, false, false, true, true, false, false, false, false
213 | 135: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
214 | 136: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
215 | 137: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
216 | ];

```

Test prompt 2

```

1 robot1l = [0: outside, 1: outside, 2: goingToDoor1, 3: goingToDoor1, 4: atDoor1, 5: insideRoom1, 6: goingToDoor3, 7:
atDoor3, 8: goingDoor3Person1, 9: rescuedPerson1, 10: rescuedPerson1, 11: rescuedPerson1, 12: rescuedPerson1, 13:
rescuedPerson1, 14: rescuedPerson1];
2 robot2l = [0: outside, 1: goingToDoor1, 2: atDoor1, 3: door1Opening, 4: door1Opening, 5: door1Opening, 6: door1Opening, 7:
door1Opening, 8: door1Opening, 9: door1Opening, 10: door1Opening, 11: door1Opening, 12: door1Opening, 13:
door1Opening, 14: door1Opening];
3 door1l = [0: closed, 1: closed, 2: closed, 3: opening, 4: open, 5: open, 6: open, 7: open, 8: open, 9: open, 10: open, 11:
open, 12: open, 13: open, 14: open];
4 door2l = [0: closed, 1: closed, 2: closed, 3: closed, 4: closed, 5: closed, 6: closed, 7: closed, 8: closed, 9: closed,
10: closed, 11: closed, 12: closed, 13: closed, 14: closed];
5 door3l = [0: open, 1: open, 2: open, 3: open, 4: open, 5: open, 6: open, 7: open, 8: open, 9: open, 10: open, 11: open,
12: open, 13: open, 14: open];
6 robot1u = [0: doNothing, 1: goToDoor1, 2: doNothing, 3: doNothing, 4: doNothing, 5: goToDoor3, 6: doNothing, 7:
goToDoor3Person1, 8: doNothing, 9: doNothing, 10: doNothing, 11: doNothing, 12: doNothing, 13: doNothing, 14:
doNothing];
7 robot2u = [0: goToDoor1, 1: doNothing, 2: openDoor1, 3: doNothing, 4: doNothing, 5: doNothing, 6: doNothing, 7: doNothing,
8: doNothing, 9: doNothing, 10: doNothing, 11: doNothing, 12: doNothing, 13: doNothing, 14: doNothing];
8 endStateAcceptAutomatonTraj = [0: waiting, 1: waiting, 2: waiting, 3: waiting, 4: waiting, 5: waiting, 6: waiting, 7:
waiting, 8: waiting, 9: accepted, 10: accepted, 11: accepted, 12: accepted, 13: accepted, 14: accepted];
9 endStateAcceptAutomatonTrTraj =
10 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
11 | 28: true, true, true, true, true, true, true, true, false, false, false, false, false, false, true
12 | 29: false, false, false, false, false, false, false, false, true, false, false, false, false, false, false
13 | 30: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
14 | 31: false, false, false, false, false, false, false, false, true, true, true, true, true, true, false
15 | 32: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
16 | ];
17 stageWhenAccepted = 9;
18 clockWhenPerson1Rescued = -0.0;
19 clockWhenPerson2Rescued = -0.0;
20 epAcceptAutomatonTraj = [0: epWaiting, 1: epWaiting, 2: epWaiting, 3: epWaiting, 4: epWaiting, 5: epWaiting, 6: epWaiting,
7: epWaiting, 8: epWaiting, 9: epWaiting, 10: epWaiting, 11: epWaiting, 12: epWaiting, 13: epWaiting, 14: epWaiting
];
21 epAcceptAutomatonTrTraj =
22 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
23 | 33: true, true, true, true, true, true, true, true, true, true, true, true, true, true, true
24 | 34: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
25 | 35: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
26 | 36: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
27 | 37: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
28 | 38: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
29 | 39: false, false, false, false, false, false, false, false, false, false, false, false, false, false, false
30 | ];
31 epAcceptStartEvent = 0;
32 epAcceptEndEvent = 0;
33 epAcceptClockWhenStart = -0.0;
34 epAcceptClockWhenEnd = 1.0;
35 epAcceptClockParameter = 1.0;
36 e = [0: -0.0, 1: 4.940656458412465e-324, 2: 1.0, 3: 2.0, 4: 2.0, 5: 2.0, 6: 2.0, 7: 6.0, 8: 6.0, 9: 11.0, 10: 11.0, 11: 11
.0, 12: 11.0, 13: 11.0, 14: 11.0, 15: 11.0];
37 c =
38 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
39 | robot1: -0.0, 1.0, 1.0, 1.0, -0.0, -0.0, 4.0, -0.0, 5.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
40 | robot2: -0.0, 1.0, 1.0, -0.0, -0.0, -0.0, 4.0, 4.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0, -0.0
41 | ];
42 co =
43 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14: 15:
44 | robot1: -0.0, -0.0, 1.0, 1.0, 2.0, 2.0, 6.0, 6.0, 11.0, 11.0, 11.0, 11.0, 11.0, 11.0, -0.0
45 | robot2: -0.0, 4.940656458412465e-324, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, -0.0
46 | ];
47 cl =
48 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
49 | robot1: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
50 | robot2: -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0
51 | ];
52 cu =
53 [ | 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14:
54 | robot1: -0.0, 1.0, 1.0, 2.0, 2.0, 2.0, 6.0, 6.0, 11.0, 11.0, 11.0, 11.0, 11.0, 11.0, -0.0
55 | robot2: 4.940656458412465e-324, 1.0, 2.0, 2.0, 2.0, 2.0, 4.0, 4.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0
56 | ];
57 Enol =

```


References

- [1] H. Li and B. C. Williams, “Generative planning for hybrid systems based on flow tubes,” in *International Conference on Automated Planning and Scheduling*, Sydney, Australia, 2008.
- [2] L. P. Kaelbling and T. Lozano-Pérez, “Integrated task and motion planning in belief space,” *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.
- [3] J. Chen, J. Li, Y. Huang, C. Garrett, D. Sun, C. Fan, A. Hofmann, C. Mueller, S. Koenig, and B. C. Williams, “Cooperative task and motion planning for multi-arm assembly systems,” *arXiv preprint arXiv:2203.02475*, 2022.
- [4] D. Wang and B. Williams, “Tburton: A divide and conquer temporal planner,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.
- [5] B. Williams and D. Wang, “Ctburton: A constraint-based generative planner with flexible time,” Massachusetts Institute of Technology, Technical Report, 2024, To Appear.
- [6] P. Yu, “Collaborative diagnosis of over-subscribed temporal plans,” Ph.D. dissertation, Massachusetts Institute of Technology, Oct. 2016. URL: <https://dspace.mit.edu/handle/1721.1/108926>.
- [7] P. Yu, “Continuous relaxation to over-constrained temporal plans,” M.S. thesis, Massachusetts Institute of Technology, 2013.
- [8] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “Progprompt: Generating situated robot task plans using large language models,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 11 523–11 530.
- [9] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” in *Conference on Robot Learning*, PMLR, 2023, pp. 287–318.
- [10] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9493–9500.
- [11] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, “Sayplan: Grounding large language models using 3d scene graphs for scalable task planning,” *arXiv preprint arXiv:2307.06135*, 2023.

- [12] S. Vemprala, R. Bonatti, A. Buckner, and A. Kapoor, “Chatgpt for robotics: Design principles and model abilities,” *Microsoft Auton. Syst. Robot. Res.*, vol. 2, p. 20, 2023.
- [13] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, “Nl2tl: Transforming natural languages to temporal logics using large language models,” *arXiv preprint arXiv:2305.07766*, 2023.
- [14] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, “Autotamp: Autoregressive task and motion planning with llms as translators and checkers,” *arXiv preprint arXiv:2306.06531*, 2023.
- [15] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman, *et al.*, “Foundation models in robotics: Applications, challenges, and the future,” *arXiv preprint arXiv:2312.07843*, 2023.
- [16] T. Léauté and B. C. Williams, “Coordinating agile systems through the model-based execution of temporal plans,” in *Proceedings of the National Conference on Artificial Intelligence*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, vol. 20, 2005, p. 114.
- [17] A. Hofmann and B. Williams, “Exploiting spatial and temporal flexibility for plan execution of hybrid, under-actuated systems,” in *AAAI 2006*, 2006.
- [18] A. Hofmann, “Robust execution of bipedal walking tasks from biomechanical principles,” 2006.
- [19] H. X. Li, “Kongming: A generative planner for hybrid systems with temporally extended goals,” 2010.
- [20] J. Chen, B. C. Williams, and C. Fan, “Optimal mixed discrete-continuous planning for linear hybrid systems,” in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021, pp. 1–12.
- [21] A. Garrido, M. Fox, and D. Long, “A temporal planning system for durative actions of pddl2. 1,” in *ECAI*, 2002, pp. 586–590.
- [22] R. P. Goldman, “Durative planning in htms.,” in *ICAPS*, 2006, pp. 382–385.
- [23] M. Fox and D. Long, “Modelling mixed discrete-continuous domains for planning,” *Journal of Artificial Intelligence Research*, vol. 27, pp. 235–297, 2006.
- [24] J. Frank, “What is a timeline,” in *Proc. of the 4th Workshop on Knowledge Engineering for Planning and Scheduling*, Citeseer, 2013, pp. 31–38.
- [25] N. Gigante, A. Montanari, M. C. Mayer, and A. Orlandini, “Complexity of timeline-based planning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, 2017, pp. 116–124.
- [26] N. Gigante, “Timeline-based planning: Expressiveness and complexity,” *arXiv preprint arXiv:1902.06123*, 2019.
- [27] O. Sankur, P. Bouyer, N. Markey, and P.-A. Reynier, “Robust controller synthesis in timed automata,” in *International Conference on Concurrency Theory*, Springer, 2013, pp. 546–560.
- [28] E. Asarin, O. Maler, and A. Pnueli, “Symbolic controller synthesis for discrete and timed systems,” in *Hybrid Systems II 2*, Springer, 1995, pp. 1–20.

- [29] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, “Controller synthesis for timed automata,” *IFAC Proceedings Volumes*, vol. 31, no. 18, pp. 447–452, 1998.
- [30] J. Frank and A. Jónsson, “Constraint-based attribute and interval planning,” *Constraints*, vol. 8, pp. 339–364, 2003.
- [31] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, *et al.*, “Aspen-automated planning and scheduling for space mission operations,” in *Space Ops*, vol. 82, 2000.
- [32] B. C. Williams and P. P. Nayak, “A reactive planner for a model-based executive,” in *IJCAI*, Citeseer, vol. 97, 1997, pp. 1178–1185.
- [33] S. H. Chung, “A decomposed symbolic approach to reactive planning,” Ph.D. dissertation, Massachusetts Institute of Technology, 2003.
- [34] A. Coles, A. Coles, M. Fox, and D. Long, “Forward-chaining partial-order planning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 20, 2010, pp. 42–49.
- [35] J. Benton, A. Coles, and A. Coles, “Temporal planning with preferences and time-dependent continuous costs,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 22, 2012, pp. 2–10.
- [36] G. Röger, P. Eyerich, and R. Mattmüller, “Tfd: A numeric temporal extension to fast downward,” *International Planning Competition (IPC), Sydney, Australia*, p. 114, 2008.
- [37] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [38] E. Fernandez-Gonzalez, E. Karpas, and B. Williams, “Mixed discrete-continuous planning with convex optimization,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.
- [39] M. H. Reeves, “Magellan: A robust executive enabling long horizon multi-agent campaigns,” Ph.D. dissertation, Massachusetts Institute of Technology, 2020.