RICE UNIVERSITY

Temporal Logic Motion Planning in Partially Unknown Environments

by

Matthew R. Maly

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
Master of Science

Approved, Thesis Committee:

Dr. Lydia E. Kavraki, Chair
Noah Harding Professor
Department of Computer Science

Dr. Moshe Y. Vardi
Karen Ostrum George Professor
Department of Computer Science

Dr. James McLurkin
Assistant Professor
Department of Computer Science

Houston, Texas
April 3, 2013

ABSTRACT

Temporal Logic Motion Planning in Partially Unknown Environments

by

Matthew R. Maly

This thesis considers the problem of a robot with complex dynamics navigating a partially discovered environment to satisfy a temporal logic formula consisting of both a co-safety formula component and a safety formula component. We employ a multi-layered synergistic framework for planning motions to satisfy a temporal logic formula, and we combine with it an iterative replanning strategy to locally patch the robot's discretized internal representation of the workspace whenever a new obstacle is discovered. Furthermore, we introduce a notion of "closeness" of satisfaction of a linear temporal logic formula, defined by a metric over the states of the corresponding automaton. We employ this measure to maximize partial satisfaction of the co-safety component of the temporal logic formula when obstacles render it unsatisfiable. For the safety component of the specification, we do not allow partial satisfaction. This introduces a general division between "soft" and "hard" constraints in the temporal logic specification, a concept we illustrate in our discussion of future work.

The novel contributions of this thesis include (1) the iterative replanning strategy, (2) the support for safety formulas in the temporal logic specification, (3) the method to locally patch the discretized workspace representation, and (4) support for partial satisfaction of unsatisfiable co-safety formulas. As our experimental results show, these methods allow us to quickly compute motion plans for robots with complex dynamics to satisfy rich temporal logic formulas in partially unknown environments.

Acknowledgements

I would first like to thank my advisers, Dr. Lydia Kavraki and Dr. Moshe Vardi, for their constant support, both intellectually and otherwise. This thesis, and the work leading up to it, would not have been possible without them challenging and encouraging me along the way.

I would also like to thank Dr. James McLurkin, the third member of my committee, for his invaluable suggestions for this work, beginning from the moment I proposed the idea to him in his office almost a year ago.

Finally, I would like to thank the members of the Physical and Biological Computing Group at Rice University, especially Devin Grady, Morteza Lahijanian, Ryan Luna, and Mark Moll, for many fruitful discussions, and for their willingness to listen to me incrementally present this work week after week.

This work was supported in part by NSF Expeditions 1139011, NSF CCF 1018798, and the Shared University Grid at Rice funded by NSF under grant EIA-0216467 and a partnership between Rice University, Sun Microsystems, and Sigma Solutions, Inc.

I dedicate this thesis to Maddy Sanders, without whom my successes in graduate school would feel aimless and less meaningful.

Contents

1 Introduction 1

1.1 Contributions . . . 3

2 Related Work 9

2.1 Classifying this Work . . . 10

2.1.1 On Partially Satisfying a Specification . . . 11

2.2 Motion Planning . . . 12

2.3 Sampling-Based Motion Planning . . . 13

2.3.1 Planning with Differential Constraints . . . 15

2.3.2 Discrete Guides for Continuous Motion . . . 18

2.4 On Logic Specifications for Robots . . . 22

2.4.1 Synthesis-Based Approaches . . . 22

2.4.2 Motion-Planning Approaches . . . 26

3 Temporal Motion Planning in Partially Unknown Environments 28

3.1 Preliminaries . . . 28

3.1.1 Motion Planning Problem with a Temporal Logic Specification 29

3.1.2 Syntactically Co-safe and Safe LTL . . . 30

3.2 Problem Description and Overall Approach . . . 34

3.2.1 Overall Approach . . . 35

3.3 Planning Framework . . . 38

3.3.1 Abstraction . . . 40

3.3.2 Initializing the Product Automaton . . . 43

3.3.3 Planning . . . 45

3.3.4 Discovering an Obstacle and Replanning . . . 49

4 Framework Implementation and Experimentation 52

4.1 Implementation . . . 52

4.2 Experiments . . . 54
4.2.1 The Office-Like Environment . . . 55
4.2.2 The Maze-Like Environment . . . 61

5 Possible Extensions **67**
5.1 Hard and Soft Constraints . . . 67
5.2 Beyond Co-safe and Safe LTL . . . 69

6 Conclusion and Future Work **73**

List of Figures

1.1 Continuum of tasks to be solved by robotic motion planning. 2
1.2 Schematic of an office building consisting of a lobby and five rooms. . 5

2.1 A roadmap in two dimensions. 14
2.2 A tree of motions in two dimensions. 15
2.3 The SyCLoP architecture. 17
2.4 A SyCLoP lead with tree of motions 19
2.5 A ML-LTL-MP lead with tree of motions 21
2.6 A bisimilar environment abstraction with example GR(1) specification. 23
2.7 An example hybrid controller 24
2.8 The receding horizon approach. 25

3.1 Multi-layered synergistic motion planning framework. 40
3.2 A DFA corresponding to a sequencing formula. 47

4.1 An office-like environment map. 57
4.2 A sample robot trajectory that satisfies a co-safe specification. 59
4.3 Various initial maps of the office-like environment. 60
4.4 Experimental data for office with formula φ_5 and varied initial maps. 62
4.5 A maze-like environment map. 63
4.6 The minimal DFA corresponding to φ_{cosafe}. 65
4.7 The minimal DFA corresponding to φ_{safe}. 66

5.1 A maze-like environment map. 70
5.2 The minimal DFA corresponding to the liveness formula ψ. 71

List of Tables

4.1 Experimental data for office with full and partial initial maps. 56
4.2 Experimental data for maze with full and partial initial maps. 65

5.1 Experimental data for maze with a partial initial map. 72

Chapter 1

Introduction

Classical motion planning solves for "A to B" movement, in which a robot is asked to move from position A to position B and to avoid obstacles along the way. Much work has been done in the robotics community to solve this problem very efficiently, often for robots with very high-dimensional state spaces and differential constraints, and in the presence of complex obstacles [13,16,28,29,31,45,46,60].

However, to one day have autonomous robots working in the presence of humans, we must go beyond this foundational step of "A to B" motion planning. One eventual dream, of course, is a helper robot that can robustly satisfy commands such as "empty the dishwasher" or "deliver medication to all patients on the third floor". One can imagine a wide continuum between this dream and the foundational "A to B" algorithms we have today, as illustrated in Figure 1.1.

Taking an incremental step along the planning continuum, we arrive at motion planning under a temporal logic specification. This is similar to classical motion

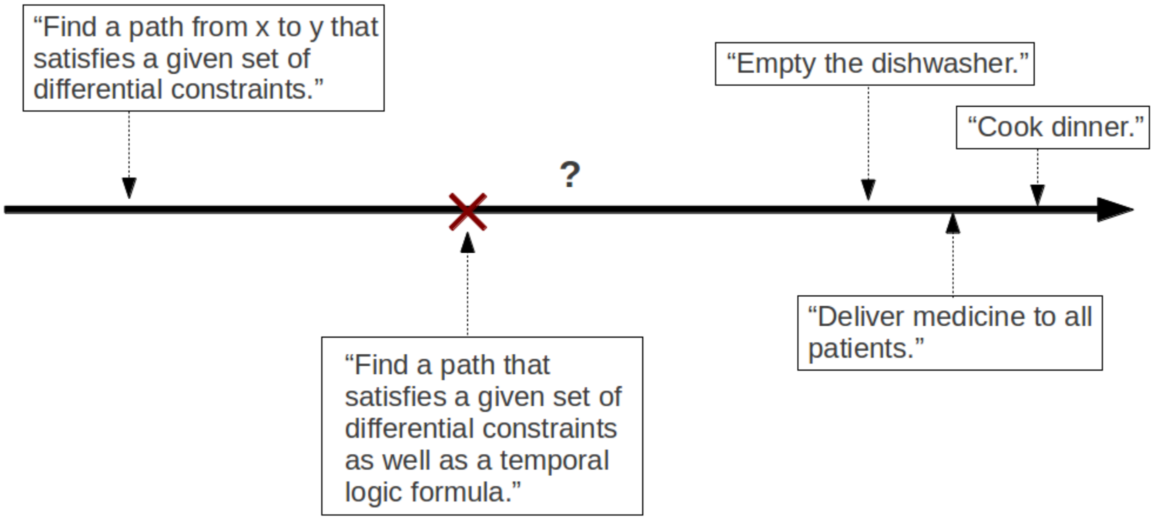

"Find a path from x to y that satisfies a given set of differential constraints."
"Empty the dishwasher."
"Cook dinner."
?
"Deliver medicine to all patients."
"Find a path that satisfies a given set of differential constraints as well as a temporal logic formula."

Figure 1.1: Continuum of tasks to be solved by robotic motion planning.

planning, except the "A to B" query is generalized to a richer set of goals that can be expressed as temporal logic formulas. Temporal logic significantly increases our task expressivity, allowing us to specify missions using connectives such as

- *disjunction*: "visit A or B",
- *coverage*: "visit A, B, and C in any order",
- *sequencing*: "visit A, B, and C in that order",
- *avoidance*: "never visit A", and
- *conditional execution*: "if condition C occurs, then perform task T", where T is some combination of the above connectives.

This thesis focuses on a framework that encodes both a task to complete and behaviors to avoid for motion planning given a linear temporal logic (LTL) specification. LTL is an instantiation of the type of expressiveness described above, and it has proven to be very popular in the robotics research community for high-level robotic control.

1.1 Contributions

Most of the existing works in motion planning with temporal logic specifications consider static workspaces with full knowledge of their maps [3–5, 61–63]. Such assumptions, however, do not usually hold in real-world scenarios. For instance, a mobile robot in a warehouse setting may not be aware of a fallen box from a shelf that has blocked an aisle, or a mobile robot in an office environment may not know about the states of the office doors before its deployment. In such scenarios, it is reasonable to assume some information about the environment (e.g., the floor plan of the warehouse or the office building), but the motion-planning framework needs to have the capability of dealing with unforeseen obstacles in the environment. With complex specifications, it is imperative to consider the cases where the environment changes [52].

Recent works in synthesis-based approaches to robotic control have begun to consider such cases (e.g., [49, 50, 68]). In these works, synthesis involves the creation of a control strategy that can account for every possible uncertainty. However, for the cases in which the number of environmental uncertainties is large, the problem becomes too complex. In these cases, it can be advantageous to adopt an iterative temporal planning approach, in which online replanning is performed when unexpected environmental features are discovered. The difference between synthesis-based approaches and the iterative temporal planning approach proposed by this thesis is discussed further in Section 2.4. In short, synthesis can theoretically support all types of unknowns, but the approach becomes computationally infeasible as the number of unknowns grows large. The iterative temporal planning approach, as proposed in [52] and presented in this thesis, can only deal with one specific type of unknown (undiscovered obstacles in the robot's workspace), but the approach can support a very high number of potential unknowns, i.e., the possibility of any shape of obstacle being discovered anywhere in the environment at any time.

Additionally, the iterative temporal planning approach poses the extra question of what to do in case components of the specification cannot be satisfied due to newly discovered obstacles in the environment. For example, the robot may discover a closed door preventing access to a region of interest which the specification requires the robot to visit, rendering the specification unsatisfiable. This gives rise to the need for a formal definition of a measure of satisfaction of a specification, a topic which is partially addressed in this thesis as well as in [52].

Consider, for instance, a janitor robot in an office building whose schematic representation is shown in Figure 1.2. The office environment consists of a lobby and five rooms, each with a door. The regions of interest in this office environment are shown as orange, purple, red, green, yellow, and brown rectangles. The first five rectangles represent desks in the office that the robot must clean, and the brown rectangle represents a region that the obstacle should avoid. An example of a motion specification is as follows:

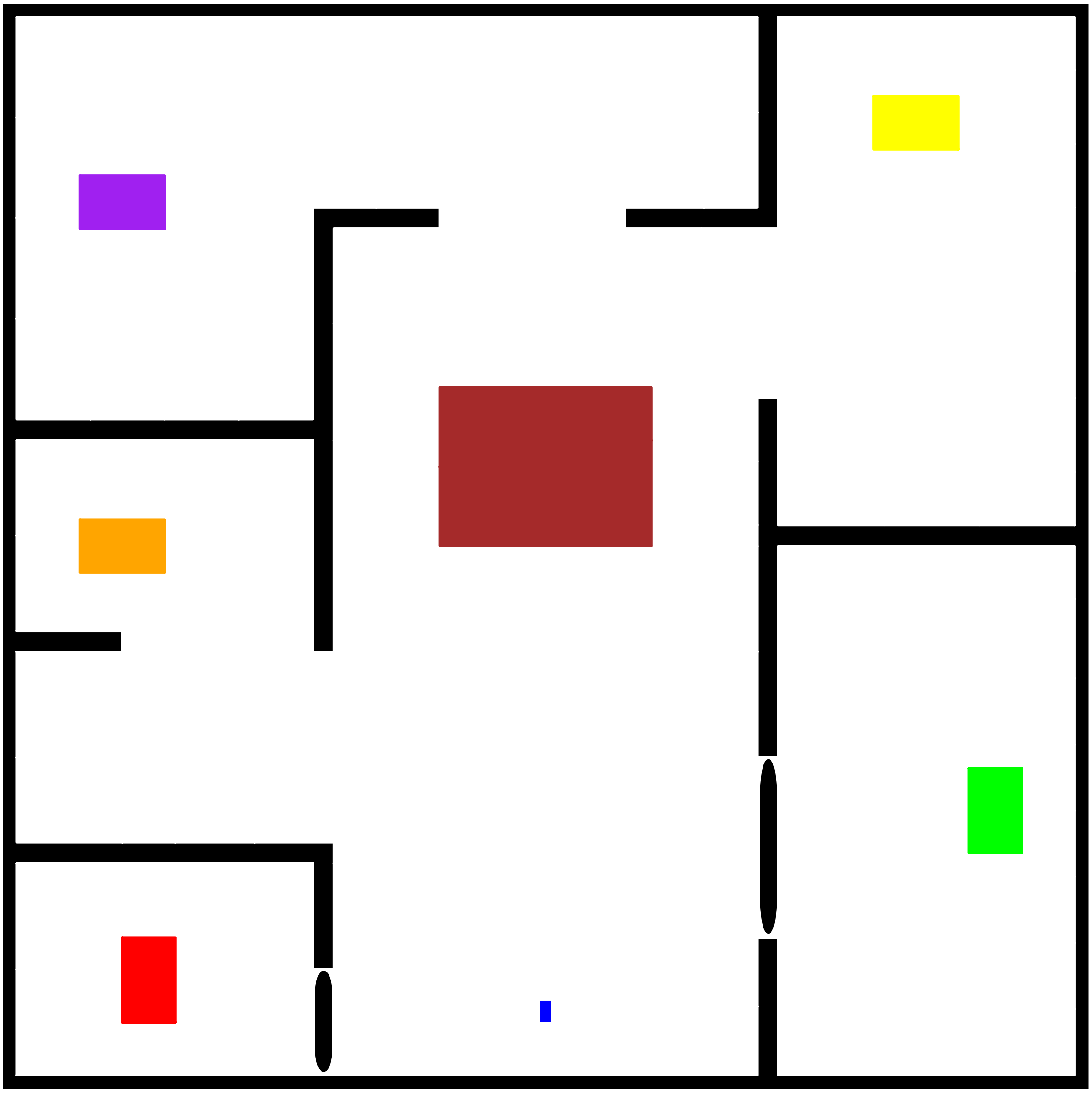


Figure 1.2: A schematic representation of an office building consisting of a lobby and five rooms. Each room has a door, of which three are open and two are closed. The robot is shown as a blue rectangle in the lobby. The regions of interest in this environment are represented by the red, orange, purple, yellow, green, and brown rectangles.

Specification 1 *Visit the red, green, orangle, purple, and yellow regions to clean the desks in any order. Do not touch the brown region.*

In this example, the robot initially has no knowledge of the state of the office doors. It discovers them as it moves in the environment. In this case, the robot will not be able to visit the red and green regions since the doors of their rooms are closed. Thus, there are components of the above specification that cannot be satisfied. Nevertheless, for such tasks, we should allow the robot to continue with the mission even if it fails to satisfy parts of the specification due to unforeseen environmental constraints. Specifically, we allow the robot to partially satisfy an unachievable task ("visit the red, green, orange, purple, and yellow regions") as long as it does not violate a safety condition ("avoid the brown region"). In this example, the robot should be expected to satisfy Specification 1 as closely as possible by visiting the orange, purple, and yellow regions and not touching the brown region.

This thesis considers such cases in the context of planning for a robot with complex dynamics to satisfy a linear temporal logic specification. We assume the robot is equipped with a range sensor, allowing it to perfectly detect obstacles within some radius ρ of its center. When a new obstacle is discovered by the robot as it moves along a trajectory, the robot performs a braking operation to come to a stop. Then the robot "patches" its internal discrete representation of its workspace to reflect the new obstacle. Finally, the robot computes a new solution trajectory and resumes execution. It is possible that the newly discovered obstacle renders components of the temporal logic specification unsatisfiable. As will be defined in Section 3.1.2, the temporal logic specification consists of a *co-safety* formula and a *safety formula*, corresponding to a set of tasks to complete and a set of behaviors to avoid, respectively. As long as the unsatisfiability of the temporal logic specification stems from an unachievable task (i.e., an unsatisfiable co-safety formula) and not a forced violation of a safety condition, we will allow planning to proceed. In such cases, the co-safety formula is satisfied as closely as possible, using a metric defined over the states of the corresponding automaton.

The novel contributions of this thesis are as follows:

- An iterative technique for replanning in the presence of unforeseen obstacles in the environment [52],
- a method to maximize the partial satisfaction of a co-safe formula [52],
- a method to identify and patch only the components of the robot's discrete representation of its workspace (called the *abstraction*) that are affected by a newly discovered obstacle, and
- an integration of safety formulas into the temporal logic planning framework.

The first two contributions, concerning unforeseen obstacles and partial satisfaction of co-safe formulas, were first presented in [52]. The latter two contributions are new to this thesis.

We have implemented the framework as part of The Open Motion Planning Library (OMPL) [17] and have tested it on a second-order car-like robot in multiple environments in which the regions of interest are known a priori.

The remainder of the thesis is organized as follows. Chapter 2 contains related work. Chapter 3 describes our robot model and the type of temporal logic that we use, and then details the problem we consider and gives an overview of our approach to solving it. In Chapter 4, we introduce our implementation of the framework and discuss the results of our experiments. In Chapter 5, we discuss possible extensions to our framework's use of temporal logic specifications. The thesis concludes with final remarks and a discussion of future work in Chapter 6.

Chapter 2

Related Work

This thesis presents a framework that uses sampling-based motion planning to compute trajectories for a robot with arbitrarily complex dynamics to satisfy a linear temporal logic specification (consisting of co-safe and safe components) in a partially unknown environment. The purpose of this chapter is to explain the related research that has been done in this area and to place this thesis in context of the related work. We begin by describing at a high level the key differences between this work and related works. We then describe all related works, beginning with a historical description of motion planning, leading to sampling-based motion planning with differential constraints and with logical specifications. We then describe synthesis-based approaches to a similar problem to the one considered in this thesis.

2.1 Classifying this Work

This work is most closely related to [3–5, 52, 61] in that we are taking a motion-planning approach instead of using synthesis. Synthesis-based approaches require knowledge of all possible environmental uncertainties [21, 34–37, 48, 50]. In real-world applications, it may not be feasible to obtain the amount of information of the environment required for synthesis. In addition, when there are too many unknowns, synthesizing a plan can be intractable. In this thesis, we propose an online iterative planning approach to deal with one specific type of unknown: undiscovered obstacles in the robot's environment. Rather than accounting for everything that can go wrong, we plan based on what we know and deal with environmental changes only when they are discovered. In other words, we plan a trajectory given the currently known state of the environment. During execution of the trajectory, if an unforeseen problem is encountered, we replan a new trajectory from the current state on-the-fly. This framework is inspired by replanning scenarios in robotics [1, 2].

A key advantage of our approach lies in the high-level structure through which we guide a low-level continuous motion planner. This high-level structure is a product of the workspace abstraction and a pair of automata that derive from the components of the temporal logic specification. The abstraction is quickly computed by a triangulation of the robot's workspace. However, the automata from the specification can be very expensive to compute [4, 44]. By keeping the automata and the abstraction separate, we prevent changes to the environment from requiring us to recompute the automata. Changes to the environment simply require modifications to the corresponding abstraction, which is an inexpensive operation to perform. This is in contrast to other works that use synthesis-based approaches to plan for robots to satisfy temporal logic specifications. In these works, typically the task specification and assumptions on the environment and the robot's dynamics must be encoded into a single hybrid controller that can be expensive to change [36, 49, 50].

Another advantage of our framework is the use of motion planning, which supports systems with any type of high-dimensional complex (possibly nonlinear) dynamics. Synthesis-based approaches deal with a restricted class of robot systems that typically involve linear dynamics. When the dynamics of the system are sufficiently complex, it is difficult (if not impossible) to synthesize provably correct controllers [5]. In general, our motion-planning framework supports arbitrary "black box" dynamics at the expense of bisimilarity, which is impossible with synthesis. Moreover, our framework's high-level structure, which we use to guide a low-level motion planner, does not depend on the continuous dynamics of the robot. The dynamics only come into play during planning.

2.1.1 On Partially Satisfying a Specification

One additional distinguishing feature of our framework is how we compute trajectories to partially satisfy the co-safe component of the specification in situations in which it is unsatisfiable. The issue of what to do when a specification is determined to be unsatisfiable has been explored before. In [64], an algorithm was defined to report a reason as to why a GR(1) LTL specification is unrealizable. The work in [32] and [33] presents a method of changing an unsatisfiable nondeterministic Büchi automaton into the “closest” satisfiable one, where all actions of the robot are represented using a finite state machine. Our approach to unsatisfiable specifications differ in that we do not change the corresponding automata; instead, we provide a simple metric to define partial satisfaction that is meaningful for many interesting scenarios.

Next, in the following sections, we will describe in detail the related works in motion planning and synthesis for robotics.

2.2 Motion Planning

Classical motion planning began with the notion of a configuration space to compute trajectories of rigid body objects through environments with obstacles [51]. The configuration space, also called the *state space*, is the space of all states that a robot can achieve. The dimension of a robot’s state space is equivalent to the number of degrees of freedom of the robot. Complete motion-planning algorithms were created to compute collision-free paths for geometric planning but did not scale well [65]. Specifically, the motion-planning problem was shown to be PSPACE-complete with respect to the numbers of degrees of freedom of the robot [10]. Early solutions include (1) cell decomposition methods that partition the state space into a connected set of convex cells and consequently do not scale well with dimension, and (2) potential fields approaches, in which the goal state is assigned an attractive force and the obstacles a repulsive force, but overcoming the issue of local minima in the state space is quite difficult [13].

2.3 Sampling-Based Motion Planning

In response to the intractability of exact solutions, much of the motion planning research community shifted its focus to sampling-based approaches, which trade completeness guarantees for tractable time complexity. Such approaches offer *probabilistic completeness*, which means that the probability that such an algorithm will find a solution (assuming one exists) approaches 1 as the algorithm spends more time on the problem. A probabilistically complete motion planner cannot in general detect if a solution does not exist [13].

Sampling-based motion planning algorithms can be roughly divided into two categories: *roadmap-based* and *tree-based*. The roadmap-based planners are best represented by `PRM`, which was also the first sampling-based motion planning algorithm in general [31]. `PRM` proceeds by sampling a large number of collision-free points from the state space, and connects each point to its nearby neighboring points with the use of a *local planner*. The resulting *roadmap* is a graph that approximates the connectivity of the free configuration space. Once a roadmap is built, it can be used for solving multiple planning queries in the future. For each pair of start and goal configurations, the states are connected into the roadmap, and then a solution path is generated using a graph search, such as Dijkstra's shortest path algorithm or A* search. Figure 2.1 contains an example of a roadmap in a two-dimensional configuration space, with the start and goal configurations connected.

The second category of sampling-based motion-planning algorithms, the tree-based planners, are meant for single-query motion planning, in which a pair of start

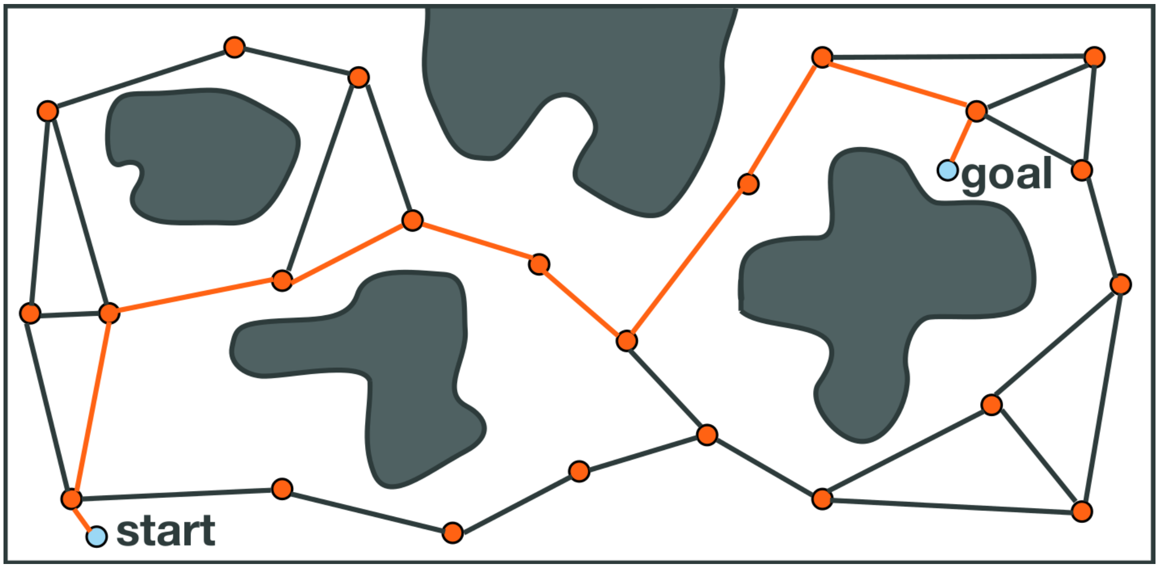


Figure 2.1: A roadmap in two dimensions.

and goal configurations are known beforehand. One-way motions are simulated as edges of a tree, rooted at the start configuration. The growth of the tree halts once the planner creates a leaf sufficiently close to the goal configuration. Following the edges back from the leaf to the root of the tree yields a solution path. Tree-based planners include the seminal `RRT` [45] and `EST` [28] planners, and their many extensions, e.g., [9, 38, 56, 66, 70], which vary primarily in (1) what areas of the tree are chosen for expansion in each step, and (2) how expansion in a given area of the tree is performed [14]. Figure 2.2 contains an example of a tree of motions in a two-dimensional configuration space. The tree is rooted at the start configuration and has successfully reached the goal configuration. For additional details, a rigorous survey of tree-based planners can be found in [15].

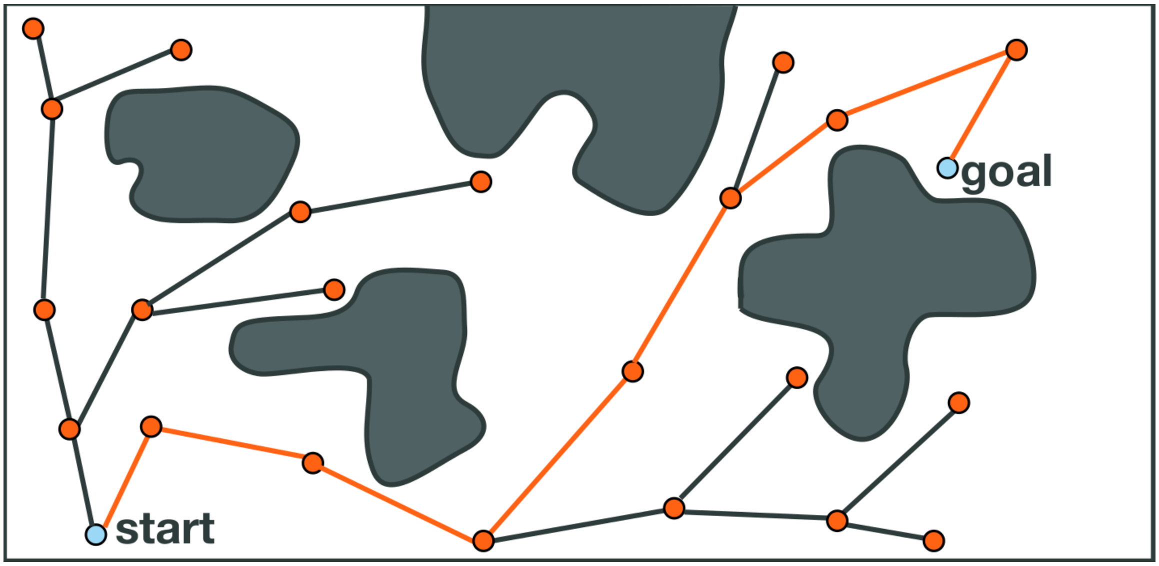


Figure 2.2: A tree of motions in two dimensions.

2.3.1 Planning with Differential Constraints

The success of sampling-based motion planning algorithms has prompted researchers to apply them to increasingly difficult problems. One class of such problems includes robotic systems with differential constraints [13]. In these systems, robots can only exhibit motions that are realized by the application of a sequence of controls. The classic motion planning problem can be generalized to incorporate robotic dynamics by including the additional requirement that the computed trajectory satisfies the differential constraints imposed by the robot's equations of motion. Many tree-based planners can easily be generalized to solve such problems, where a tree state q not only holds a pointer to its parent state $p(q)$ but also stores the necessary controls to realize a motion from $p(q)$ to q. Typically these controls are not computed as a function of $p(q)$ and q; instead, a control is generated (often randomly) and applied to

$p(q)$ to obtain its child state q. For roadmap-based planners, solving such problems is less natural. Specifically, it is in general very difficult to construct a local planner capable of generating a control to connect two specific nearby states.

Planning with differential constraints is much more difficult than with simple geometric constraints. In general, it is not known if the problem is even decidable [12]. With a complex enough problem (say, more than five degrees of freedom), planning with simple algorithms such as RRT or EST can quickly become infeasible. More advanced sampling-based motion-planning algorithms are needed to more efficiently guide the search toward a solution, instead of naively covering the state space. Three such algorithms that have been shown to be successful are KPIECE [16], PDST [41], and SyCLoP [60]. All three of these planners use low-dimensional projections to guide the tree of motions. KPIECE chooses where to expand its tree of motions by considering the tree's coverage of a space determined by some low-dimensional projection [16]. PDST dynamically subdivides a projected subspace of the state space in order to estimate coverage without the use of a metric [41]. SyCLoP, an extension of an older planner called DSLX [58,59] creates sequences of neighboring regions (called high-level *guides* or *leads*) through a discretization of the workspace along which a low-level planner guides a tree of motions [60]. In general, SyCLoP can operate over a discretization of any low-dimensional projected subspace of the robot's state space, but the choice of projection can be problem-specific and does not yield significant benefits to performance [53]. As the low-level planner gathers information of the success or failure of a given lead, this information is sent back up to the high-level planner to inform the creation of future leads. The architecture diagram in Figure 2.3 illustrates SyCLoP's information flow. SyCLoP has been shown to give performance improvements of up to two orders of magnitude over efficient implementations of RRT and EST [60].

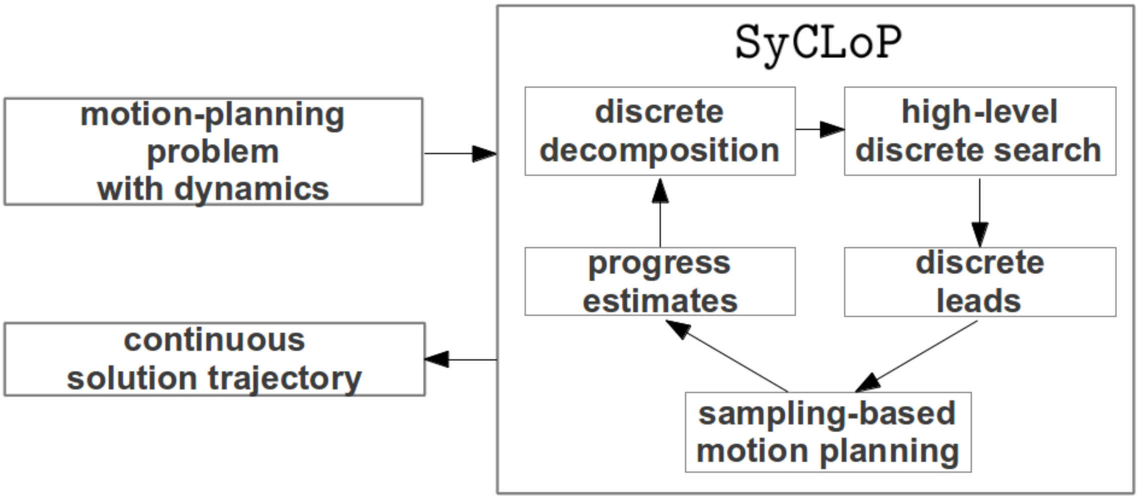

SyCLoP
motion-planning problem with dynamics
discrete decomposition
high-level discrete search
progress estimates
discrete leads
continuous solution trajectory
sampling-based motion planning

Figure 2.3: The SyCLoP architecture, taken from [53].

Figure 2.4 illustrates how SyCLoP guides a tree of motions along a lead. Here, a robot with second-order car-like dynamics begins at the bottom center of an office-like environment. Its goal is to move to the purple region in the top-left of the environment. SyCLoP accepts as input a geometric partition (often called a *decomposition* or *abstraction*) of the workspace, which in this example is a triangulation that respects obstacles. To quickly reach the goal, SyCLoP calculates a sequence of neighboring regions, beginning with the triangle containing the start state and ending with a triangle in the goal region. Then, a low-level sampling-based tree planner is executed along the lead. In Figure 2.4, the lead is denoted by red arrows, and the states of the tree are denoted by blue points. In the case of this example, the low-level planner was able to quickly find a solution trajectory with only one lead. With more difficult problems, SyCLoP will guide the low-level tree along a lead for some time Δt, use information gathered from that exploration to compute a new lead, and then repeat the process. Leads are computed using a shortest-path graph search, where the vertices of the graph correspond to regions in the abstraction, and the edges correspond to adjacencies between regions. An edge between two regions r_1 and r_2 is assigned an edge weight of the form

$$w(r_1, r_2) = \frac{\textsc{numsel}(r_1) \cdot \textsc{numsel}(r_2)}{\textsc{cov}(r_1) \cdot \textsc{cov}(r_2) \cdot \textsc{vol}(r_1) \cdot \textsc{vol}(r_2)}, \tag{2.1}$$

where $\textsc{numsel}(r_i)$ is the number of times SyCLoP has expanded the tree of motions in region r_i, $\textsc{cov}(r_i)$ is the number of tree vertices associated with region r_i (an estimate of coverage), and $\textsc{vol}(r_i)$ is the free area of the workspace contained by region r_i. There are many variants of edge weight formulas for planners similar to SyCLoP (e.g., [4, 5, 60–62]). In general, edge weight functions that incorporate at least the information used in (2.1) seem to perform well. A rigorous survey of edge weight functions remains a topic of future work.

2.3.2 Discrete Guides for Continuous Motion

We now introduce the planning framework that is the subject of this thesis. As discussed in the previous section, the SyCLoP framework has been shown to successfully guide a tree of motions along a high-level lead. In its original conception, the high-level lead is simply a sequence of regions from the region containing the start state to the region containing the goal state; i.e., the framework solves the classic

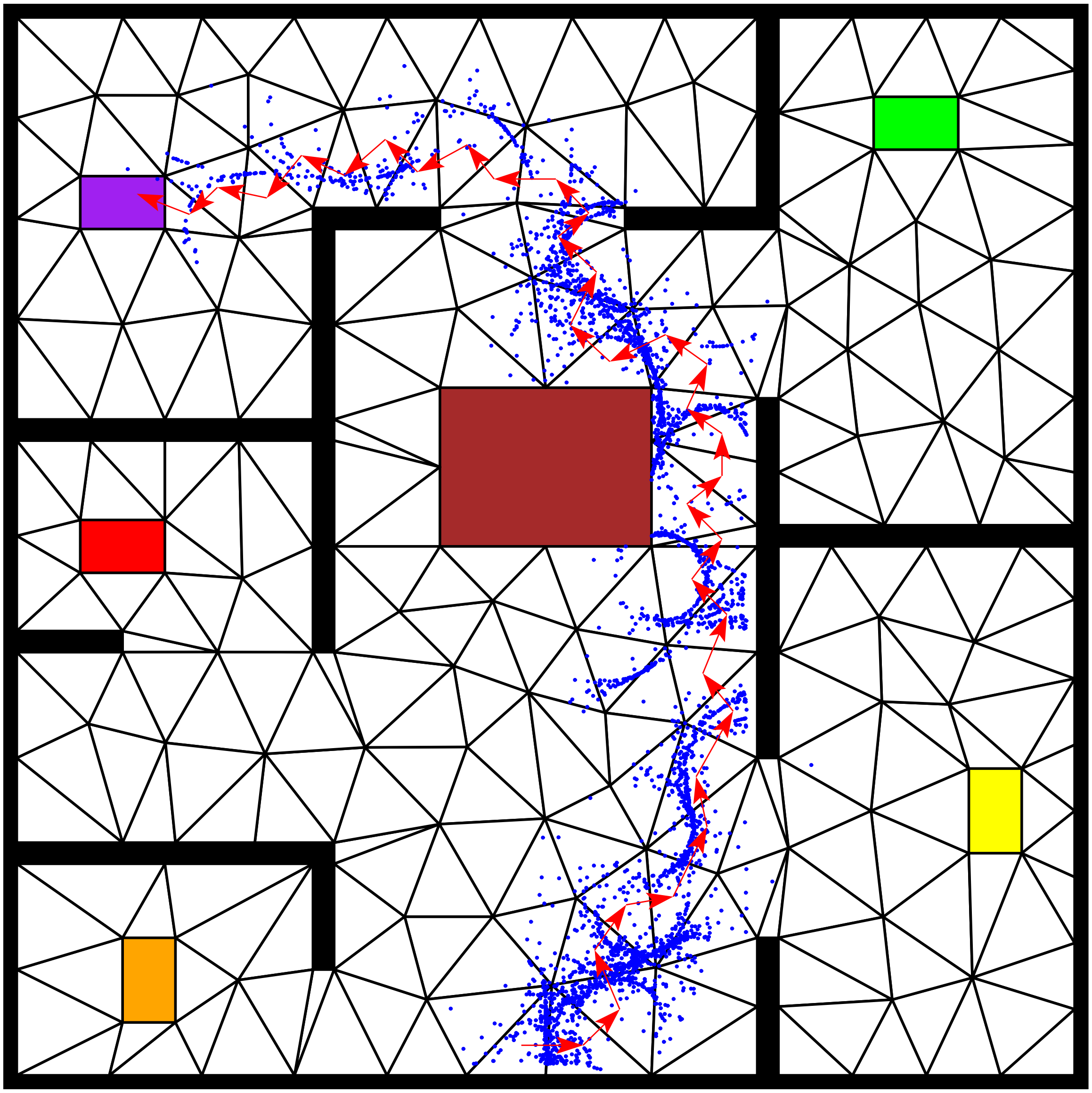


Figure 2.4: A lead (denoted by red arrows) and tree of motions (denoted by blue points) from one run of `SyCLoP`, where the robot's start state is at the bottom center of the environment, and the goal state is the purple region.

"A to B" planning problem. To leverage this framework for planning with temporal logic constraints, SyCLoP was extended to form the *multi-layered LTL motion planner* (ML-LTL-MP) [4]. ML-LTL-MP guides a low-level tree of motions along a high-level lead as before. What has changed is that the high-level lead is no longer simply a sequence of regions through a workspace discretization. Instead, the workspace discretization is combined with the automaton corresponding to the temporal logic specification to form a *product automaton*. The start state of this product automaton is the pairing (d_0, z_0) of the discrete workspace region d_0 containing the robot's start state, and the automaton's start state z_0. A product automaton state (also called a *high-level state*) is *accepting* if its component automaton state is accepting. A high-level lead is any sequence of high-level states that ends in an accepting state. In Figure 2.5, a second-order car begins at the bottom center of an office-like environment. Its goal is to satisfy a temporal logic specification representing the task "visit the purple and green regions in any order" (often called a *coverage* formula). As before, ML-LTL-MP computes a lead that satisfies the specification, and a tree of motions is guided along that lead until a solution trajectory is found.

ML-LTL-MP has been shown to improve performance over RRT by orders of magnitude when computing trajectories to satisfy temporal logic specifications [4, 5, 62]. The framework we present in this thesis is an extension of ML-LTL-MP; we will discuss it in more detail in Chapter 3.

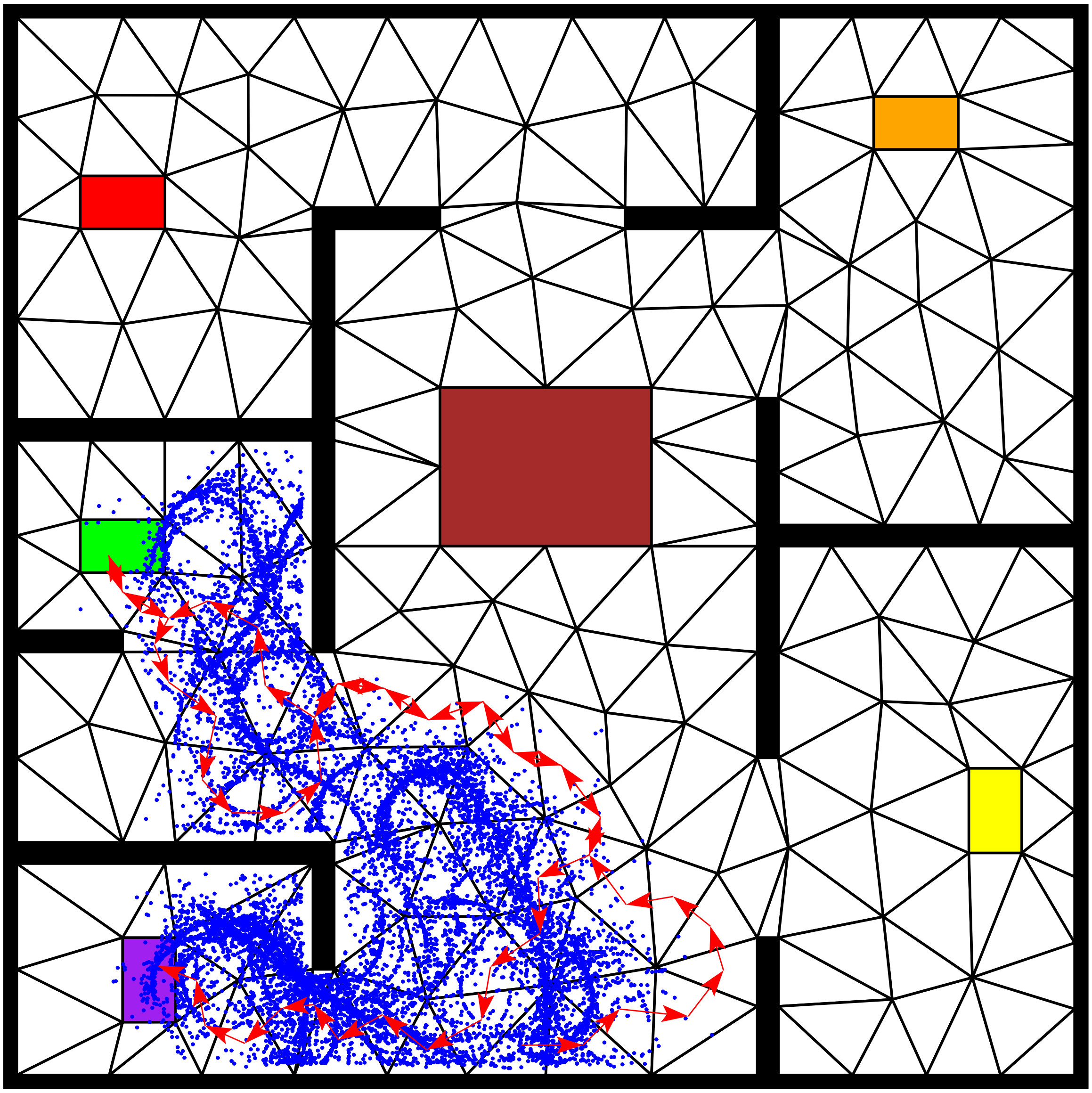


Figure 2.5: A lead (denoted by red arrows) and tree of motions (denoted by blue points) from one run of `SyCLoP`, where the robot's start state is at the bottom center of the environment, and the goal is to satisfy the task "visit the purple and green regions in any order".

2.4 On Logic Specifications for Robots

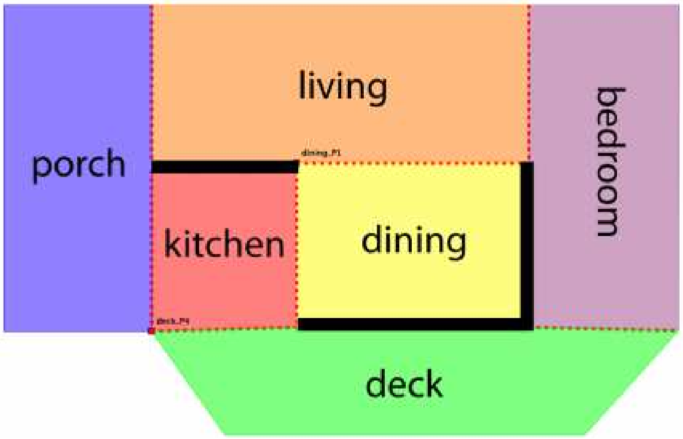
Our framework is not the only one in the literature for computing robot motions to satisfy logical specifications more complex than "A to B" reachability. Much work has been done toward the problem of planning for robotic systems to satisfy high-level temporal logic specifications. We divide all such work into two categories: (1) synthesis-based approaches and (2) motion-planning approaches.

2.4.1 Synthesis-Based Approaches

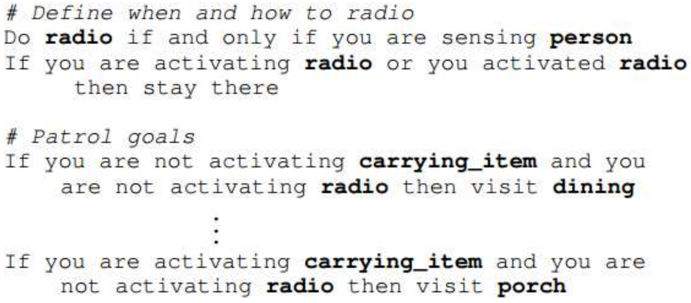
Synthesis-based approaches for controlling robotic systems to satisfy high-level specifiactions require strong assumptions on the robot's dynamics and the existence of bisimilar controllers to move between workspace regions [21, 34–37, 48, 50]. Moreover, a special subset of LTL, called GR(1) (*generalized reactivity formulas of rank 1*) is typically used in such approaches [7, 57]. Generally, a GR(1) formula is of the form

$$\bigwedge_i \varphi_i \rightarrow \bigwedge_j \psi_j,$$

where φ_i and φ_j are LTL formulas that are representable by deterministic Büchi automata. The left side of the implication is meant to encode all possible environment behaviors, which includes not only adjacency information of regions in the environment, but also any environment features that can be sensed by the robot. The right side of the implication encodes all robot behavior. Figure 2.6(b), taken from [22], contains an excerpt of a GR(1) specification written in structured English. This specification corresponds to the environment abstraction given in Figure 2.6(a). This abstraction must be bisimilar to the dynamics model of the robot; that is, given any two adjacent regions in the abstraction (for example, the living room and the bedroom), there must exist a controller that is guaranteed to take the robot from any point in the first region to somewhere within the second region [36].



(a)



```
# Define when and how to radio
Do radio if and only if you are sensing person
If you are activating radio or you activated radio
     then stay there

# Patrol goals
If you are not activating carrying_item and you
    are not activating radio then visit dining
                    ⋮
If you are activating carrying_item and you are
    not activating radio then visit porch
```

(b)

Figure 2.6: (a) a bisimilar abstraction of a robot's environment; (b) a GR(1) specification written in structured English; both figures taken from [22].

Given a robot model, a GR(1) specification, and a bisimilar environment abstraction, a provably correct hybrid controller can be generated as a state machine that encodes the robot actions necessary to satisfy the task [36]. An example of such a hybrid controller, taken from [34], is contained in Figure 2.7.

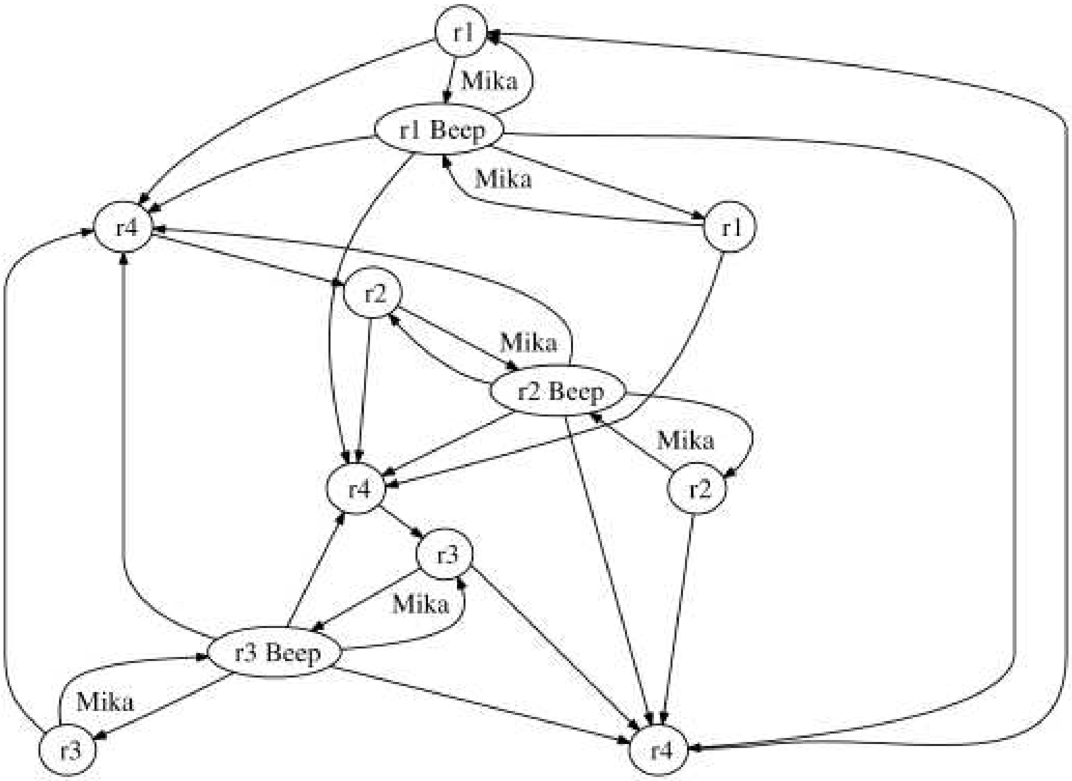


Figure 2.7: An example hybrid controller synthesized from a GR(1) specification, taken from [34].

The synthesis of the hybrid controller requires time and space polynomial in the size of the reachable state space of the system. This is often called the *state explosion problem* [37]. One way to address this problem is to use a coarser abstraction, which requires a stronger assumption on the robot's controllers, an assumption that is often difficult to guarantee. Other suggestions include receding horizon techniques, in which the set of desired liveness properties encoded in the GR(1) specification is partitioned into a sequence of short-horizon plans, which can replace a very large hybrid controller with a connected sequence of small hybrid controllers [74]. Figure 2.8 contains such a sequence, taken from [75]. Here, the sets $\mathcal{W}_4, \ldots, \mathcal{W}_0$ form a partition of the states of the hybrid controller, and each v_i is an individual state. The initial state of the system may be any one of the states in $\mathcal{W}_4 = \{v_1, v_2, v_3, v_4\}$. The goal state of the system is v_{10}. The partitioning scheme $\mathcal{W}_4, \ldots, \mathcal{W}_0$ is chosen so that the original GR(1) specification of the system is split into a sequence of realizable short-horizon specifications. Unfortunately, the partition and the path horizon must be manually chosen in this approach.

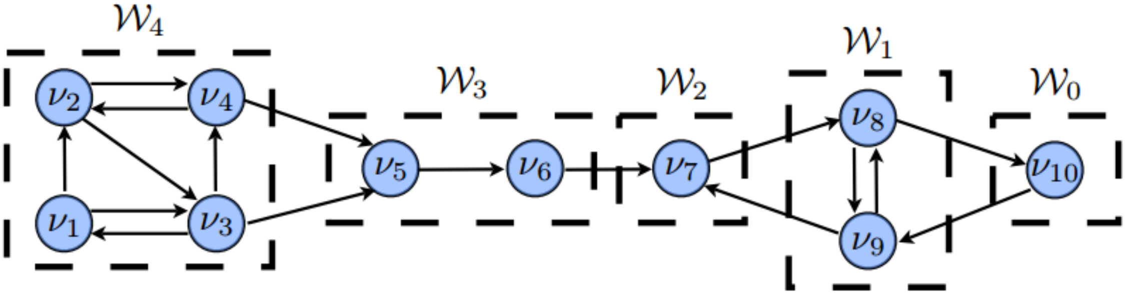


Figure 2.8: A sequence of hybrid controllers, illustrating the receding horizon approach. The sequence is defined by the sets $\mathcal{W}_4, \ldots, \mathcal{W}_0$ which form a partition of the states of the hybrid controller. Each v_i is an individual state of the controller. $\mathcal{W}_4$ contains the initial states, and v_{10} is the goal state. Taken from [75].

Work has also been done to address the issue of controller uncertainty in this context, modeling the robot as a Markov decision process [19, 42, 43].

In an Unknown or Changing Environment

The synthesis techniques in the works discussed so far require a predictable environment [21, 22, 34–37]. In addition, much work has been done on synthesis for problems in which the predictability of the environment cannot be guaranteed. The issue of synthesis from high-level specifications in an unknown or dynamic environment has been studied both for abstract systems (e.g., [6]) and specifically for robotics (e.g., [11, 33, 49, 50, 68]). If the geometry of the environment changes, whether due to an unknown region becoming reachable [68] or a known region becoming unreachable [49, 50], then the hybrid controller must be updated to incorporate the change. As global resynthesis of the hybrid controller is expensive, there exist approaches to locally patch the controller to incorporate the changes in less time [49, 50]. In such works, the states and transition edges of the hybrid controller corresponding to changing region must be identified, removed, and then replaced with states and transitions that reflect the new properties of the region. Initial work in this area has shown that patching the hybrid controller can still require significant time to complete, in some cases requiring as much time as resynthesizing the entire hybrid controller from scratch [49, 50].

2.4.2 Motion-Planning Approaches

Our framework, which has been extended from `ML-LTL-MP` [4, 5] is one more step in a long chronology of planning methods for robotic systems and hybrid systems to satisfy co-safe LTL specifications [3–5, 52, 61–63]. However, this chronology is not the only work that has been done on motion-planning approaches for satisfying temporal logic formulas. For general robot models with nonlinear dynamics, static workspaces, and temporal goals, a motion-planning approach has been proposed to solve the problem using deterministic μ-calculus specifications [30]. In that work, the authors propose a planner called the *rapidly-exploring random graph* (`RRG`), which can be seen as a tree-based motion planner with cycles in the graph. The authors assume the existence of a local steering method to maneuver between nearby points. To compute a trajectory that satisfies a given deterministic μ-calculus specification, the `RRG` algorithm incrementally builds a graph in the state space until it contains a path that satisfies the specification. Though deterministic μ-calculus, is more expressive than LTL and easy to model-check, it is very difficult to write. Moreover, the specification does not affect how the graph is built - such an approach (often called a *monitor-based* approach) has been shown to not perform well with a high-dimensional state space and a complex specification [4].

Chapter 3

Temporal Motion Planning in Partially Unknown Environments

3.1 Preliminaries

In Section 3.1.1, we formally define the problem statement for motion planning in a partially unknown environment with a temporal logic specification. Section 3.1.2 defines the two subsets of LTL that we use. Section 3.2 describes at a high level our approach to solving the problem, and Section 3.3 describes our planning framework in detail.

3.1.1 Motion Planning Problem with a Temporal Logic Specification

In this thesis, we consider a general mobile robot with complex dynamics in a partially unknown environment and a temporal logic specification consisting of co-safety formula and safety formula components. We assume that the robotic system consists of

1. $\mathcal{Q} \subset \mathbb{R}^n$, a bounded n-dimensional state space, an element of which completely determines the robotic system's state,

2. $\mathcal{U} \subset \mathbb{R}^c$, a bounded c-dimensional control space consisting of control variables that can be applied to the system to change its state,

3. $\textsc{Flow} : \mathcal{Q} \times \mathcal{U} \rightarrow \dot{\mathcal{Q}}$, a differential equation that captures the system's constraints,

4. $\textsc{Valid} : \mathcal{Q} \rightarrow \{0, 1\}$, a boolean function describing whether a state is valid (used for collision avoidance),

5. $q_{\text{init}} \in \mathcal{Q}$, a start state for the system,

6. $\mathcal{W} \subset \mathbb{R}^2$, a bounded 2-dimensional representation of the *workspace* in which the robot resides,

7. $\textsc{Proj} : \mathcal{Q} \rightarrow \mathcal{W}$, a projection function to extract the workspace location of the robot given its full state,

8. Sense : $\mathcal{Q} \rightarrow \{0, 1\}$, a sensing function that returns 1 if a previously unknown obstacle is discovered at a given state,

9. $\Pi = \{p_1, \ldots, p_k\}$, a set of atomic propositions, and

10. $L : \mathcal{W} \rightarrow 2^{\Pi}$, a state-labeling function assigning to each robot system state a set of atomic propositions that hold true at that state.

3.1.2 Syntactically Co-safe and Safe LTL

We use syntactically co-safe and syntactically safe LTL to write the specifications of robotic tasks. Co-safe LTL will be used to encode tasks for the robot to achieve, and safe LTL will be used to encode behaviors for the robot to avoid. Their syntax and semantics are defined below.

Definition 1 (Co-safe Syntax) *Let* $\Pi = \{p_1, \ldots, p_k\}$ *be a set of boolean atomic propositions. A* syntactically co-safe *LTL formula over* Π *is inductively defined as follows:*

$$\varphi := p \,|\, \neg p \,|\, \varphi \vee \varphi \,|\, \varphi \wedge \varphi \,|\, \mathcal{X}\varphi \,|\, \varphi \mathcal{U} \varphi \,|\, \mathcal{F}\varphi$$

where $p \in \Pi$, $\neg$ *(negation),* $\vee$ *(disjunction), and* $\wedge$ *(conjunction) are boolean operators, and* $\mathcal{X}$ *("next"),* $\mathcal{U}$ *("until"), and* $\mathcal{F}$ *("eventually") are temporal operators.*

Definition 2 (Safe Syntax) *A* syntactically safe *LTL formula over* Π *is inductively* *defined as follows:*

$$\varphi := p \,|\, \neg p \,|\, \varphi \vee \varphi \,|\, \varphi \wedge \varphi \,|\, \mathcal{X}\varphi \,|\, \mathcal{G}\varphi$$

where $p \in \Pi$, $\neg$ *(negation),* $\vee$ *(disjunction), and* $\wedge$ *(conjunction) are boolean operators, and* $\mathcal{X}$ *("next") and* $\mathcal{G}$ *("always") are temporal operators.*

Definition 3 (Semantics) *The semantics of syntactically co-safe and safe LTL formulas are defined over infinite traces over* 2^{Π}*. Let* $\sigma = \{\tau_i\}_{i=0}^{\infty}$ *with* $\tau_i \in 2^{\Pi}$ *be an infinite trace, and define* $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$ *and* $\sigma^i = \tau_i, \tau_{i+1}, \ldots$*. Then* σ_i *is a prefix of the trace* σ*, and* σ^i *is a suffix of* σ*. The notation* $\sigma \models \varphi$ *indicates that* σ *satisfies formula* φ *and is inductively defined as follows.*

- $\sigma \models \pi$ *if* $\pi \in \tau_0$*;*
- $\sigma \models \neg\pi$ *if* $\pi \notin \tau_0$*;*
- $\sigma \models \varphi_1 \vee \varphi_2$ *if* $\sigma \models \varphi_1$ *or* $\sigma \models \varphi_2$*;*
- $\sigma \models \varphi_1 \wedge \varphi_2$ *if* $\sigma \models \varphi_1$ *and* $\sigma \models \varphi_2$*;*
- $\sigma \models \mathcal{X}\varphi$ *if* $\sigma^1 \models \varphi$*;*
- $\sigma \models \varphi_1 \mathcal{U} \varphi_2$ *if* $\exists k \geq 0$*, s.t.* $\sigma^k \models \varphi_2$*, and* $\forall i \in [0, k)$*,* $\sigma^i \models \varphi_1$*;*
- $\sigma \models \mathcal{F}\varphi$ *if* $\exists k \geq 0$*, s.t.* $\sigma^k \models \varphi$*.*
- $\sigma \models \mathcal{G}\varphi$ *if* $\forall k \geq 0$*,* $\sigma^k \models \varphi$*.*

An important property of syntactically co-safe LTL formulas is that, even though they have infinite-time semantics, finite traces are sufficient to satisfy them. Similarly, finite traces are sufficient to violate syntactically safe LTL formulas. Hence, we can capture desired robot behavior as a pair of co-safe and safe LTL specifications, where the co-safe component describes tasks for the robot to complete, and the safe component describes behaviors to avoid. We then say that a trajectory *satisfies* the pair of specifications if it satisfies the co-safe component and does not violate the safe component. This combination of co-safe and safe LTL formula components will allow us to describe many rich types of robotic tasks which can be realized in a finite time horizon in a safe manner.

To evaluate robotic trajectories against LTL formulas, we use deterministic finite automata [27]. A deterministic finite automaton (DFA) is given by a tuple $(Z, \Sigma, \delta, z_0, F)$, where

- Z is a finite set of states,
- $\Sigma = 2^{\Pi}$ is the input alphabet, where each input symbol is a truth assignment to the propositions in Π,
- $\delta : Z \times \Sigma \to Z$ is the transition function,
- $z_0 \in Z$ is the initial state, and
- $F \subseteq Z$ is the set of accepting states.

A *run* of a DFA $\mathcal{A}$ is a sequence of states $w = w_0 w_1 \ldots w_n$, where $w_0 = z_0$ and $w_i \in \mathcal{A}.Z$ for $i = 1, \ldots, n$. A run w is called an *accepting run* if $w_n \in \mathcal{A}.F$.

From a syntactically co-safe LTL formula φ_{cosafe}, a DFA $\mathcal{A}_{\varphi_{\text{cosafe}}}$ can be constructed that accepts precisely all of the formula's satisfying finite traces [40]. Each input symbol to $\mathcal{A}_{\varphi_{\text{cosafe}}}$ (and, in general, to any DFA generated from a logical specification) is a set $\sigma \in 2^{\Pi}$ of propositions that are currently true in the system. Similarly, from a syntactically safe LTL formula φ_{safe}, a DFA $\mathcal{A}_{\neg\varphi_{\text{safe}}}$ can be constructed that accepts precisely all of the formula's violating finite traces [40]. To accept precisely all of the finite traces that do not violate φ_{safe}, we flip the acceptance condition of this DFA to obtain $\neg\mathcal{A}_{\neg\varphi_{\text{safe}}}$, which we minimize [27] and refer to simply as $\mathcal{A}_{\text{safe}}$. Specifically, given $\mathcal{A}_{\neg\varphi_{\text{safe}}} = (Z, \Sigma, \delta, z_0, F)$, we define

$$\mathcal{A}_{\varphi_{\text{safe}}} = (Z, \Sigma, \delta, z_0, Z \setminus F)$$

and then minimize $\mathcal{A}_{\varphi_{\text{safe}}}$. The DFA $\mathcal{A}_{\varphi_{\text{safe}}}$ accepts a finite trace w if and only if there exists an infinite trace extension to w that satisfies φ_{safe}; that is, the language of $\mathcal{A}_{\varphi_{\text{safe}}}$ is given by

$$\mathcal{L}\left(\mathcal{A}_{\varphi_{\text{safe}}}\right) = \left\{u \in \Sigma^* \mid \exists v \in \Sigma^\omega \text{ such that } uv \models \varphi_{\text{safe}}\right\}.$$

Here Σ^ω denotes the set of all infinite traces over the alphabet Σ.

Throughout this thesis, we will loosely say that a finite trace w "satisfies the safety formula φ_{safe}" to mean that w has an infinite trace extension that satisfies φ_{safe}, or equivalently, that w does not violate φ_{safe}.

If a finite robot trajectory corresponds to a trace accepted by $\mathcal{A}_{\varphi_{\text{safe}}}$, then that trajectory does not violate the safety condition defined in φ_{safe}. Similarly, if a finite robot trajectory corresponds to a trace accepted by $\mathcal{A}_{\varphi_{\text{cosafe}}}$, then that trajectory correctly completes the task defined in φ_{cosafe}. We use our framework to generate robot trajectories that are accepted by both $\mathcal{A}_{\varphi_{\text{safe}}}$ and $\mathcal{A}_{\varphi_{\text{cosafe}}}$.

3.2 Problem Description and Overall Approach

In this thesis, we consider a mobile robot with complex and possibly non-linear dynamics moving in an environment to satisfy a pair of specifications $\varphi = (\varphi_{\text{cosafe}}, \varphi_{\text{safe}})$. We assume that while the robot has full information of the propositional regions and their locations in the environment, it has only partial a priori knowledge of the obstacles of the environment. This assumption is motivated by scenarios such as the one described in Section 1.1, in which the robot has a blueprint of a floor of an office building, but small details in the environment are unknown, such as the specific locations of furniture or the statuses of doors.

Due to possible unknown obstacles in the environment, the satisfaction of the specification cannot be guaranteed. Nevertheless, we do not want the robot to abort the mission if it realizes that fragments of the specification cannot be met. Instead, we require the robot to satisfy the co-safe component of the specification as closely as possible. We envision many scenarios where this can be an advantageous approach (e.g., the janitor robot example in Chapter 1). We formally define and discuss the definition of satisfying a specification as closely as possible below and in Section 3.3.3. We now focus on the following problem.

Problem: *Given a partially unknown environment and a task specification* *expressed as a syntactically co-safe LTL formula φ_{cosafe} and a syntactically safe LTL formula φ_{safe} over Π, find a robot motion plan that does not violate φ_{safe} and satisfies φ_{cosafe} as closely as possible.*

We assume that the robot has partial a priori knowledge of the obstacles in its workspace. In other words, some of the obstacles could be unknown before the deployment of the robot. We also assume that the robot can detect an unknown obstacle when it comes within some proximity of it. This is represented by the function SENSE in the definition of our planning problem in Section 3.1.1. In practice, SENSE can be viewed as a range sensor with a fixed radius ρ, as is commonly seen in related work [2].

3.2.1 Overall Approach

We employ a multi-layered synergistic framework [4, 61] to solve the motion planning problem by using the initial knowledge of the workspace. The framework consists of three main layers: a high-level search layer, a low-level search layer, and a synergy layer that facilitates the interaction between the high-level and the low-level search layers (see Figure 3.1). The high-level planner uses an abstraction of the workspace and the specification φ to suggest high level plans. The low-level planner uses the dynamics of the robotic system and the suggested high-level plans to explore the state space for feasible solutions. In our work, the low-level layer is a sampling-based planner and does not assume the existence of a controller [60].

To satisfy a specification in a partially undiscovered environment, an iterative high-level planner is employed. Every time an unknown obstacle is encountered, the high-level planner modifies the coarse high-level plan online by accounting for the geometry of the discovered obstacle, the path traveled to that point, and the remaining segment of the specification that is yet to be satisfied. This replanning is achieved in four steps.

1. First, a “braking” operation is applied to prevent the robot from colliding with the newly discovered obstacle. We assume that the robot’s sensing radius is sufficiently large to ensure that the robot will not collide with the obstacle.

2. Second, the workspace abstraction is “patched” to reflect the new changes to the environment. These changes are propagated to the product automaton. All feasibility estimates for high-level states and edges that are not affected by the changed portion of the workspace are preserved.

3. Finally, the path traveled by the robot so far is mapped onto the updated product automaton. A new satisfying plan is generated as a continuation of the explored portion of the old plan.

Thus, the robot does not need to return to its starting point every time it encounters an unknown environmental feature. Moreover, the robot’s progress in satisfying the specification is preserved. This iterative motion-planning framework is discussed in detail in Section 3.3.

Recall that from $\varphi = (\varphi_{\text{cosafe}}, \varphi_{\text{safe}})$, a pair of DFAs, $\mathcal{A}_{\varphi_{\text{cosafe}}}$ and $\mathcal{A}_{\varphi_{\text{safe}}}$, can be constructed that accept all of the satisfying finite traces for φ_{cosafe} and φ_{safe}, respectively, as discussed in Section 3.1.2 [40, 61]. We use these DFAs to design a satisfying high-level plan. We also utilize $\mathcal{A}_{\varphi_{\text{cosafe}}}$ to define a metric to measure the "distance-to-satisfaction" of a specification in cases in which the co-safe component φ_{cosafe} is unsatisfiable. This measure is used to produce a high-level plan that completely satisfies φ_{safe} (i.e., does not violate φ_{safe}) and satisfies φ_{cosafe} as closely as possible. The definition of this metric is described in Section 3.3.3.

In general, a contingency maneuver can be used instead of a "braking" operation as the first step of the approach. Our framework is by no means limited to a stopping maneuver, and the exploration for the "best" contingency plan is left for future work. A description of general contingency plans can be found in [2, 23].

Moreover, it is important to note that our method of generating a new high-level plan is fast. This is for the following two reasons:

1. We are not recomputing the two DFAs, which do not need to change since the specification does not change following the discovery of an obstacle.

2. We generate the workspace abstraction by triangulating the two-dimensional environment, which has been shown to be computationally inexpensive [5] (a fact we will verify in our experimental results in Chapter 4. Our method to "locally patch" the changed region of the workspace abstraction essentially boils down to a retriangulation of the smaller portion.

For instance, the computation time for recomputing the workspace abstraction for the janitor robot example moving in the office environment shown in Figure 1.2 is on the order of a hundredth of a second on a modern PC.

3.3 Planning Framework

In this section, we describe our iterative planning framework, which consists of three main layers: a high-level planner, a low-level search layer, and a synergy layer as shown in Figure 3.1. The high-level planner generates a set of coarse satisfying plans by searching over a structure called a *product automaton* (Section 3.3.2). This structure is the product of the following three structures:

1. The discrete abstraction $\mathcal{M}$ of the robot's workspace (see Section 3.3.1),
2. the DFA $\mathcal{A}_{\text{cosafe}}$ corresponding to the formula φ_{cosafe}, and
3. the DFA $\mathcal{A}_{\text{safe}}$ corresponding to the formula φ_{safe}, as defined in detail in Section 3.1.2.

Each of these plans is a sequence of states of the product automaton. Since the two DFAs in the product automaton run on input propositions defined in the workspace and respected by the boundaries of $\mathcal{M}$, a high-level plan can be completely described by the corresponding underlying sequence of regions of $\mathcal{M}$.

The low-level search layer produces continuous trajectories that follow a satisfying high-level plan. This is achieved by expanding a sampling-based motion tree in the direction of a suggested high-level plan in the workspace. The synergy layer facilitates the two-way interaction between the high-level and the low-level search layers (see Sections 3.3.2 and 3.3.3). Algorithm 3.3.1 contains the framework pseudocode; it relies on subroutines detailed in Algorithms 3.3.2, 3.3.3, 3.3.4, 3.3.5, and 3.3.6. In the following sections, we describe these algorithms in detail.

Algorithm 3.3.1 Framework for planning for a robotic system with an LTL specification in a partially unknown environment

Input: A motion planning problem $\textsc{mpp} = (\mathcal{Q}, \mathcal{U}, \textsc{Flow}, \textsc{Valid}, q_{\text{init}}, \mathcal{W}, \textsc{Proj}, \textsc{Sense}, \Pi, L)$,
as described in Section 3.1.1),
a set of initially known obstacles $O \subset \mathcal{W}$,
a pair $\varphi = (\varphi_{\text{cosafe}}, \varphi_{\text{safe}})$ of co-safe and safe LTL formulas defined over $\textsc{mpp}.\Pi$,
and a time bound $t_{\max}$.

Output: Returns `true` if successful in moving the robot through the workspace to satisfy φ_{safe} and satisfy φ_{cosafe} as closely as possible; returns `false` otherwise.

```
 1: $\mathcal{M} \leftarrow \textsc{ComputeAbstraction}(\mathcal{W}, O, \Pi, L)$
 2: $\mathcal{A}_{\varphi_{\text{cosafe}}} \leftarrow \textsc{ComputeMinDFA}(\varphi_{\text{cosafe}}, \mathcal{W}, L)$
 3: $\mathcal{A}_{\varphi_{\text{safe}}} \leftarrow \textsc{ComputeMinDFA}(\varphi_{\text{safe}}, \mathcal{W}, L)$
 4: $\mathcal{P} \leftarrow \textsc{ComputeProduct}(\mathcal{M}, \mathcal{A}_{\varphi_{\text{cosafe}}}, \mathcal{A}_{\varphi_{\text{safe}}}, \Pi, L)$
 5: $\{x_i\}_{i \geq 0} \leftarrow \textsc{Plan}(\textsc{mpp}, O, \mathcal{P}, t_{\max})$
 6: $t_{\text{plan}} \leftarrow$ time spent by Plan in line 5
 7: $t_{\max} \leftarrow t_{\max} - t_{\text{plan}}$
 8: $j \leftarrow 1$
 9: while $j < |\{x_i\}|$ do
10:     Move robot from state $x_{j-1}.s$ to state $x_j.s$
11:     if $\textsc{Sense}(x_j.s) = 1$ then
12:         Apply braking operation to reach stopped robot state $s'$
13:         $q_{\text{init}} \leftarrow s'$
14:         Add discovered obstacle $o_{\text{new}}$ to $O$
15:         $\mathcal{P} \leftarrow \textsc{PatchProduct}(\mathcal{P}, \mathcal{Q}, \mathcal{W}, o_{\text{new}}, \Pi, L)$
16:         $\{x_i\}_{i \geq 0} \leftarrow \textsc{Plan}(\textsc{mpp}, O, \mathcal{P}, t_{\max})$
17:         if Plan was unsuccessful then
18:             return false
19:         $t_{\text{plan}} \leftarrow$ time spent by Plan in line 16
20:         $t_{\max} \leftarrow t_{\max} - t_{\text{plan}}$
21:         $j \leftarrow 1$
22:     $j \leftarrow j + 1$
23: return true
```

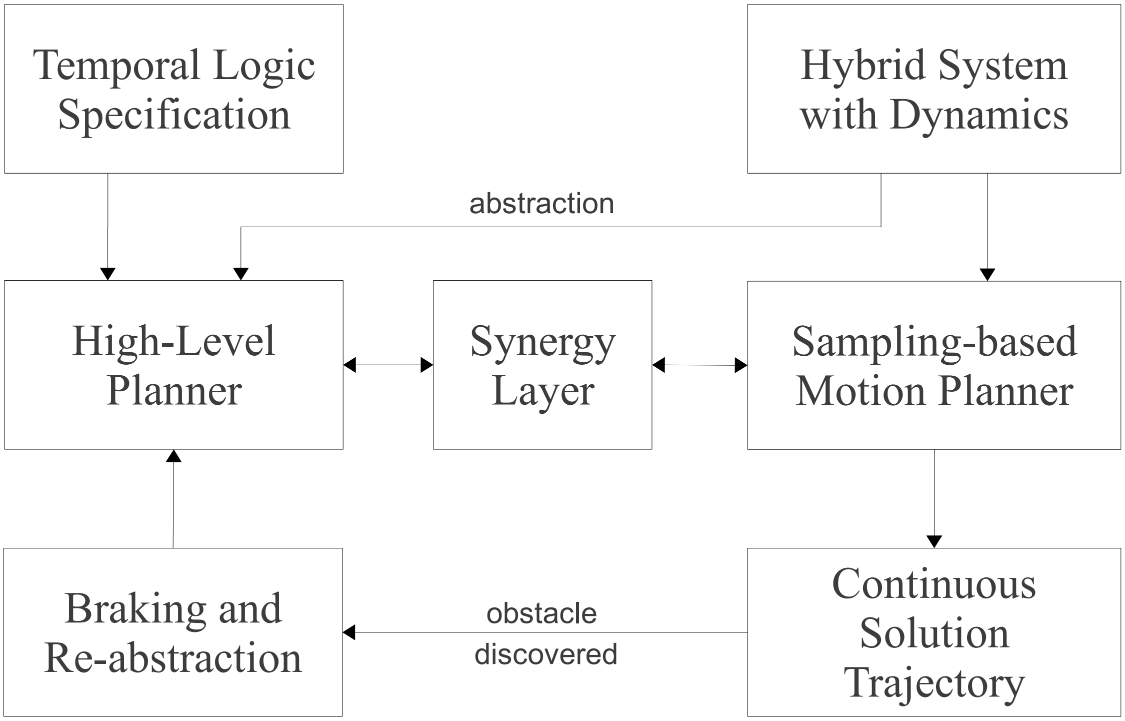

Temporal Logic Specification
Hybrid System with Dynamics
abstraction
High-Level Planner
Synergy Layer
Sampling-based Motion Planner
Braking and Re-abstraction
obstacle discovered
Continuous Solution Trajectory

Figure 3.1: Multi-layered synergistic motion planning framework.

3.3.1 Abstraction

To produce a high-level plan, we first, in line 1 of Algorithm 3.3.1, abstract the workspace $\mathcal{W}$ to a discrete model $\mathcal{M} = (D, d_0, \rightarrow_D, \Pi, L_D)$, where D is a set of discrete regions in $\mathcal{W}$, $d_0 \in D$ is the initial region, $\rightarrow_D \subseteq D \times D$ is the transition relation, and $L_D : D \rightarrow 2^{\Pi}$ is a labeling function. We refer to the model $\mathcal{M}$ as the *abstraction* of the workspace. In this work, $\mathcal{M}$ is constructed as a geometry-based conforming Delaunay triangulation of $\mathcal{W}$ that respects the propositional regions and the boundaries of the known obstacles [69]. We construct the transition relation $\rightarrow_D$ to reflect the geometric adjacencies between regions in $\mathcal{W}$.

Algorithm 3.3.2 Plan: Temporal planning algorithm

Input: A motion planning problem $\textsc{mpp} = (\mathcal{Q}, \mathcal{U}, \textsc{Flow}, \textsc{Valid}, q_{\text{init}}, \mathcal{W}, \textsc{Proj}, \textsc{Sense}, \Pi, L)$,
a set of known obstacles $O \subset \mathcal{W}$,
a product automaton $\mathcal{P}$,
and a time bound $t_{\max}$.

Output: Returns a sequence of triplets, each containing a robot system state, control, and corresponding high-level state, representing a system trajectory that satisfies the specification. Reports an error and aborts if no such trajectory could be found within time $t_{\max}$.

```
$\mathcal{T} \leftarrow \textsc{InitializeTree}(q_{\text{init}})$
while $\textsc{Time Elapsed} < t_{\max}$ do
    $K = ((d_1, z_1^c, z_1^s), \ldots, (d_k, z_k^c, z_k^s)) \leftarrow \textsc{ComputeLead}(\mathcal{P}, q_{\text{init}})$
    $C \leftarrow \textsc{ComputeAvailableCells}(K)$
    $v \leftarrow \textsc{Explore}(H, W, O, \mathcal{T}, C, K, \mathcal{P}, \Delta t)$
    if $v \neq$ NULL then
        Follow $v$.parent to construct trajectory $\{x_i\}_i$
        return $\{x_i\}_i$
Report unsuccessful and exit
```

Algorithm 3.3.3 ComputeLead: Subroutine to compute high-level guides

Input: A product automaton $\mathcal{P}$ and a starting high-level state $(d_0, z_0^c, z_0^s) \in \mathcal{P}$.

Output: Returns a lead, which is a sequence of high-level states beginning with the given start (d_0, z_0^c, z_0^s) and ending with a state that is accepting in $\mathcal{A}_{\varphi_{\text{safe}}}$ and as close as possible to an accepting state in $\mathcal{A}_{\varphi_{\text{cosafe}}}$.

```
$S \leftarrow \{(d, z^c, z^s) \in \mathcal{P} \mid z^s$ is accepting in $\mathcal{P}.\mathcal{A}_{\varphi_{\text{safe}}}\}$
$F \leftarrow \arg\min_{(d,z^c,z^s) \in S} (\textsc{DistFromAcc}(z, \mathcal{P}.\mathcal{A}_{\varphi_{\text{cosafe}}}))$
Run Dijkstra's all-pairs shortest-path algorithm on $\mathcal{P}$ with source $(d_0, z_0^c, z_0^s)$; store
parent map parent and weight map weight
$(d_g, z_g^c, z_g^s) \leftarrow \arg\min_{(d,z^c,z^s) \in F} \{$weight$[(d, z^c, z^s)]\}$
Construct lead $K = ((d_0, z_0^c, z_0^s), \ldots, (d_g, z_g^c, z_g^s))$ using parent map
return $K$
```

It should be noted that the initial construction of $\mathcal{M}$ is based on the initial knowledge of the environment map. As the robot discovers unknown obstacles, the map is updated and $\mathcal{M}$ is patched to reflect the new workspace information. Given that

Algorithm 3.3.4 Explore: Tree-exploration subroutine

Input: A motion planning problem $\textsc{mpp} = (\mathcal{Q}, \mathcal{U}, \textsc{Flow}, \textsc{Valid}, q_{\text{init}}, \mathcal{W}, \textsc{Proj}, \textsc{Sense}, \Pi, L)$,
a set of known obstacles $O \subset \mathcal{W}$,
a tree of motions $\mathcal{T}$,
a set of available high-level states C,
a lead K,
a product automaton $\mathcal{P}$,
and an exploration time Δt.

Output: Returns a tree vertex that reaches a goal high-level state if such a vertex was found; returns `NULL` otherwise.

```
while Time Elapsed $< \Delta t$ do
    $(d, z^c, z^s) \leftarrow C$.sample()
    $v \leftarrow \textsc{SelectAndExtend}(\mathcal{T}, \textsc{mpp}, (d, z^c, z^s), O, \mathcal{P})$
    if $v.z^c \neq \varnothing$ and $v.c^s \neq \varnothing$ then
        if $v.z^c$.isAccepting() and $v.z^s$.isAccepting() then
            return $v$
        if $(v.d, v.z^c, v.z^s) \in L \setminus C$ then
            $C \leftarrow C \cup \{(v.d, v.z^c, v.z^s)\}$
return NULL
```

Algorithm 3.3.5 PatchProduct: Subroutine to locally patch product automaton given a newly discovered obstacle

Input: A product automaton $\mathcal{P}$,
a robot state space representation $\mathcal{Q}$,
a workspace representation $\mathcal{W}$,
a newly discovered obstacle o_{new},
a set of atomic propositions Π, and
a state labeling function $L : Q \to 2^{\Pi}$.

Output: Returns a patched version of $\mathcal{P}$ that respects the newly discovered obstacle.

```
$R \leftarrow \textsc{GetIntersectingRegions}(\mathcal{P}.\mathcal{M}, o_{\text{new}})$
$(V, E) \leftarrow \textsc{ComputeBoundary}(R)$
$N \leftarrow \textsc{DecomposePortion}(V, E, \mathcal{Q}, \Pi, L)$
$\mathcal{P} \leftarrow \textsc{PatchAbstraction}(\mathcal{P}, R, N)$
return $\mathcal{P}$
```

Algorithm 3.3.6 ComputeBoundary: Subroutine to compute the boundary of a connected set of triangles in the workspace abstraction

Input: A connected set R of triangles in the workspace.
Output: Returns a planar straight-line graph representing the boundary

```
$B \leftarrow \varnothing$
for each triangle $T \in R$ do
    $B \leftarrow B \cup \{u, v \in T \mid$ triangle edge $(u, v)$ faces an obstacle or a triangle not in $R\}$
return $B$
```

this method is based on a triangulation of a two-dimensional space, patching the abstraction is fast. Furthermore, we initially assume transitions between all adjacent partitions of the workspace are realizable even though the dynamics of the robot may prevent some transitions. This does not create a problem in our planning framework because the synergistic framework will bias its discrete search against unrealizable transitions. In fact, one of the advantages of our planning framework is that it does not require a bisimilar abstraction as was described in Section 2.4.1 and therefore allows for inexpensive and fast construction of an approximate abstraction model.

Next, we describe how $\mathcal{M}$ is utilized in generating satisfying high-level plans.

3.3.2 Initializing the Product Automaton

The structure we use to guide the tree of system trajectories is a product automaton, which is computed as

$$\mathcal{P} = \mathcal{M} \times \mathcal{A}_{\varphi_{\text{cosafe}}}.Z \times \mathcal{A}_{\varphi_{\text{safe}}}.Z.$$

In lines 2-3 of Algorithm 3.3.1, we compute the minimal DFAs $\mathcal{A}_{\varphi_{\text{cosafe}}}$ and $\mathcal{A}_{\varphi_{\text{safe}}}$ corresponding to the formulas φ_{cosafe} and φ_{safe}, respectively, as defined in Section 3.1.2 [40,44]. Though each translation can require time doubly exponential with respect to the number of propositions in the formula, we only compute each DFA once, and so the translations can be seen as an offline step. Then, in line 4 of Algorithm 3.3.1, we compute the product automaton $\mathcal{P}$. We refer to elements of the product automaton $\mathcal{P}$ as *high-level states*. The product automaton $\mathcal{P}$ is a directed graph in which there exists an edge from high-level state (d_1, z_1^c, z_1^s) to (d_2, z_2^c, z_2^s) if and only if

1. d_1 and d_2 are adjacent in $\mathcal{M}$,

2. $\mathcal{A}_{\varphi_{\text{cosafe}}}.\delta\left(z_1^c, \mathcal{M}.L_D(d_2)\right) = z_2^c$, and

3. $\mathcal{A}_{\varphi_{\text{safe}}}.\delta\left(z_1^s, \mathcal{M}.L_D(d_2)\right) = z_2^s$,

where $\mathcal{A}_{\varphi_{\text{cosafe}}}.\delta$ is the deterministic transition function for $\mathcal{A}_{\varphi_{\text{cosafe}}}$, and $\mathcal{A}_{\varphi_{\text{safe}}}.\delta$ is the deterministic transition function for $\mathcal{A}_{\varphi_{\text{safe}}}$. We call a high-level state $(d, z^c, z^s) \in \mathcal{P}$ an *accepting* state (or a *goal* state) if z^c is an accepting state in $\mathcal{A}_{\varphi_{\text{cosafe}}}$ and z^s is an accepting state in $\mathcal{A}_{\varphi_{\text{safe}}}$.

For each high-level state $(d, z^c, z^s) \in \mathcal{P}$, we assign a weight defined by

$$w(d, z^c, z^s) = \frac{(\textsc{cov}(d, z^c, z^s) + 1) \cdot \textsc{vol}(d)}{\max\{\textsc{DistFromAcc}(z^c), \textsc{DistFromAcc}(z^s)\} \cdot (\textsc{numsel}(d, z^c, z^s) + 1)^2} \tag{3.1}$$

where $\textsc{cov}(d, z^c, z^s)$ is the number of tree vertices associated with (d, z^c, z^s) (an estimate of coverage). and $\textsc{vol}(d)$ is the area of the workspace corresponding to the abstraction state d, and $\textsc{numsel}(d, z^c, z^s)$ is the number of times (d, z^c, z^s) has been selected for tree expansion in line 2 of Algorithm 3.3.4. DistFromAcc(z^c) is the minimum distance from automaton state z^c to an accepting state in $\mathcal{A}_{\varphi_{\text{cosafe}}}$. Similarly, DistFromAcc(z^s) is the minimum distance from automaton state z^s to an accepting state in $\mathcal{A}_{\varphi_{\text{safe}}}$.

Finally, to each directed edge $e = (h_1, h_2)$ between high-level states $h_1, h_2 \in \mathcal{P}$, we assign the weight

$$w(e) = \frac{1}{w(h_1) \cdot w(h_2)} \tag{3.2}$$

The estimates in (3.1) and (3.2) have been shown to work well in previous work [4]. In general, a weighing scheme that incorporates more than just number-of-edge distance is useful to promote expansion in unexplored areas (i.e., where cov and numsel are both small) and to discourage expansion in areas where attempts at exploration have repeatedly failed (i.e., where numsel $\gg$ cov).

3.3.3 Planning

Once the product automaton has been computed, line 5 of Algorithm 3.3.1 computes a trajectory for the system that completely satisfies the safe formula φ_{safe} and satisfies the co-safe formula φ_{cosafe} as closely as possible. The details of this approach are given in Algorithm 3.3.2. Many details are similar to the framework discussed in past works [3–5]. We differ from them by (1) locally patching the product automaton and replanning new trajectories in light of newly discovered obstacles in Algorithm 3.3.1, by (2) supporting specifications that include not only co-safe formulas but also safe formulas, and by (3) partially satisfying an unsatisfiable specification when computing a lead in Algorithm 3.3.3.

The core loop of our planning algorithm is shown in lines 3, 4, and 5 of Algorithm 3.3.2. The subroutine ComputeLead in Algorithm 3.3.3 creates leads that reach as close as possible to an accepting high-level state. Each lead computed in line 3 is a suggested sequence of contiguous high-level states through which Explore attempts to guide the tree of motions.

Measure of Satisfiability We present a measure of satisfiability that uses the graph-based distance to an accepting state in the DFA $\mathcal{A}_{\varphi_{\text{cosafe}}}$. Each high-level state (d, z^c, z^s) is annotated with the graph-based distance value $\textsc{DistFromAcc}(z^c)$ corresponding to the automaton state $z^c \in \mathcal{A}_{\varphi_{\text{cosafe}}}.Z$. Our framework computes trajectories that end in a high-level state (d_g, z_g^c, z_g^s) such that

1. z_g^s is accepting in $\mathcal{A}_{\varphi_{\text{safe}}}$, and

2. $\textsc{DistFromAcc}(z_g^c)$ is minimized.

If φ_{cosafe} is satisfiable in the current environment, then $\textsc{DistFromAcc}(z_g^c) = 0$, i.e., (d_g, z_g^c, z_g^s) is an accepting state. On the other hand, if φ_{cosafe} is unsatisfiable, then z_g^c is as close as possible to accepting state in $\mathcal{A}_{\varphi_{\text{cosafe}}}$. In all cases, our framework requires that φ_{safe} is satisfiable and should abort in cases in which φ_{safe} is determined to be unsatisfiable. In many cases, there are multiple candidate high-level states that tie under the DistFromAcc metric over the co-safe automaton states. To break ties, we choose the high-level state with minimal edge-weight distance from the starting high-level state, using the edge-weight function defined in (3.2).

The function DistFromAcc is an intuitive measure on the co-safe automaton that translates to a reasonable high-level plan for many formulas that we have encountered, such as the example specification in Section 1.1. For such a specification, a trajectory that minimizes DistFromAcc takes the robot to all reachable regions of interest, while a non-optimal trajectory with respect to DistFromAcc would miss some reachable regions.

There exist other specifications in which our method of minimizing DistFromAcc does not necessarily yield the most intuitive plans. For example, consider the co-safe formula to visit regions R_1, R_2, and R_3 in that specific order:

$$\psi = \mathcal{F}\Big(R_1 \wedge \mathcal{F}\left(R_2 \wedge \mathcal{F}R_3\right)\Big).$$

The formula ψ is often called a *sequencing* formula. The minimal DFA corresponding to ψ is illustrated in Figure 3.2. If the region R_1 is inaccessible, then our approach of minimizing DistFromAcc would yield a plan for the robot to not perform any tasks at all. Specifically, since the first task of the specification, to visit R_1, is impossible, the closest the framework can get to an accepting state in the automaton is the

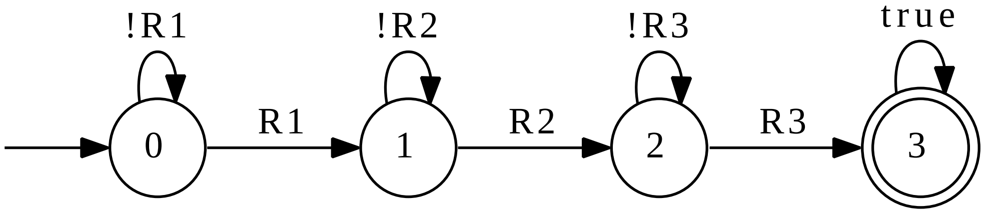


Figure 3.2: A DFA corresponding to a sequencing formula. Cases in which the propositional region R_1 is inaccessible demonstrate a weakness of our partial satisfaction approach using DistFromAcc.

initial state. In many situations, a more reasonable approach would be for the robot to "skip" the edge in the DFA corresponding to the first task, and to then visit R_2 and R_3 in order. Our current approach does not support such approximations. In a sense, our approach only allows for "skipping" of automaton edges only if they are at the end of a finite trace and conclude at an accepting state.

The topic of "approximating" temporal properties is a subject of ongoing research. Generally, it requires making the satisfaction relation quantitative rather than qualitative. For example, the satisfaction value can be an arbitrary lattice element rather than a boolean value; cf. [39]. In addition, the authors in [73] describe a synthesis algorithm to minimize quantitative satisfaction error given a set of contradictory specifications.

Guiding the Low-Level Tree Planner The subroutine COMPUTEAVAILABLECELLS in line 4 of Algorithm 3.3.2 creates a set of high-level states from the current lead that are nonempty (i.e., there exist vertices in the tree of motions that are annotated with these high-level states). To promote progress, we favor high-level states that are closest to the accepting state of the lead. Specifically, moving backwards along the lead, for each nonempty high-level state (d, z^c, z^s) we encounter, we add (d, z^c, z^s) to the set C of available high-level states and then quit early with probability 0.5. By quitting the process early with probability 0.5, we are biasing expansion toward the areas of the tree that have made the most progress along the lead, and therefore have made the most progress completing the task specification.

The subroutine EXPLORE, given in Algorithm 3.3.4, corresponds to the low-level search layer of our framework. This function promotes tree expansion in high-level states from the set C. In line 2 of Explore, a high-level state (d, z^c, z^s) is sampled with probability

$$\frac{w(d, z^c, z^s)}{\sum_{(d', z^{c\prime}, z^{s\prime}) \in C} w(d', z^{c\prime}, z^{s\prime})}.$$

Then, in line 3, a low-level tree planner attempts to create a new tree vertex corresponding to both a robot state s that maps to abstraction state d and a trajectory from the tree root that maps to automaton states z^c and z^s in $\mathcal{A}_{\varphi_{\text{cosafe}}}$ and $\mathcal{A}_{\varphi_{\text{safe}}}$, respectively. Any tree-based motion planner can be used in this step; in our implementation, we are using an `EST`-like approach [28].

If z^c and z^s are accepting states in their respective automata, then v is returned as the endpoint of a solution trajectory, which is constructed by Plan in line 7 (z^c can just be as close as possible to an accepting state if φ_{cosafe} is unsatisfiable). Otherwise, if the new vertex v corresponds to a newly reached high-level state that is in the current lead, then the high-level state is added to the set of available cells in line 8 of Explore to be considered in future iterations. We make no attempt in Plan to smooth or shorten the continuous solution trajectory. Shortening a trajectory to satisfy both differential constraints and a logical specification remains a topic of future work.

3.3.4 Discovering an Obstacle and Replanning

Once a system trajectory that satisfies φ is computed, we begin moving the robot along the trajectory. At each state in the trajectory, we query the robot's range sensor in line 11 of Algorithm 3.3.1. We assume that the robot's range sensor checks for obstacles within radius ρ of the center of the robot and reports a polygonal model of any previously unknown obstacle that it finds. If no new obstacles are discovered along the trajectory, then the robot reaches the final state of the planned trajectory and stops, having completed its mission. If an obstacle is discovered by the range sensor from some state s along the trajectory, then we apply a braking operation to the robot to reach some stopped state s' in line 12 of Algorithm 3.3.1. The braking operation should respect the dynamics of the system. In the general case, the robot should perform a contingency maneuver to avoid the newly discovered obstacle [1,23]. The radius ρ of the range sensor is assumed to be large enough for the braking or contingency maneuver to safely be performed. Once the braking maneuver is complete, we patch the portion of the discrete abstraction $\mathcal{M}$ that intersects the new obstacle by calling the subroutine PatchProduct defined in Algorithm 3.3.5, and we obtain an updated instance of $\mathcal{M}$ that ignores all known obstacles. After patching the discrete abstraction, PatchProduct patches the corresponding elements of $\mathcal{P}$. The PatchProduct routine operates in four steps, given in lines 1 through 4 of Algorithm 3.3.5:

1. Compute the set R of workspace regions of $\mathcal{M}$ that intersect with the new obstacle.

2. Compute the exterior boundary of the set R as a planar straight-line graph (V, E), using Algorithm 3.3.6.

3. Compute a new triangulation N of the section of the workspace enclosed by

(V, E).

4. Insert the new triangulation N into the abstraction $\mathcal{M}$ and propagate the changes to the product automaton $\mathcal{P}$.

After the product automaton has been patched, we replan a trajectory from s' in line 16 of Algorithm 3.3.1, following the same planning approach described in Section 3.3.3. Once a new trajectory is found by the planner, we resume moving the robot from s' along the new trajectory. It is important to note that only the high-level states of $\mathcal{P}$ that intersect with the new obstacle are replaced, and their incident edge weights are lost and recomputed in the next planning iteration. All other high-level states and edge weights in $\mathcal{P}$ are retained. Specifically, the counters represented by COV and NUMSEL in (3.2) for high-level states are do not intersect with the new obstacle are not reset to 0.

Chapter 4

Framework Implementation and Experimentation

To test our approach, we have created two experiments for a second-order car to explore an office-like environment and a maze-like environment. The full map of the office is shown in Figure 4.1(a). The full map of the maze is shown in Figure 4.5(a). For the office and maze environments, we will experiment with different combinations of co-safe and safe formula specifications.

We first briefly discuss the implementation of our framework.

4.1 Implementation

We have implemented our framework and experiments in C++ using the Open Motion Planning Library (OMPL) [17]. The main components of the implementation are the following.

1. *World* - an assignment of boolean values to propositions. A *World* can be partially restrictive. For example, $\{p_0, \neg p_3\}$ is a *World* in which p_0 is true, p_3 is false, and the other propositions can have any value. Our notion of a *World* is similar to a *truth assignment* in propositional logic.

2. *Propositional Decomposition* - a geometry-using triangulation of the workspace that ignores obstacles and respects propositional regions. Each triangle is annotated with the values of the propositional regions to which it belongs, if any; in other words, each triangle in the decomposition has a corresponding *World*.

3. *Automaton* - a deterministic finite automaton with a single start state and any number of accepting states. An *Automaton* runs over sequences of *World*s.

4. *Product Graph* - a Cartesian product of a *Propositional Decomposition* and an *Automaton*. A *Product Graph* is referred to as a product automaton in this thesis. Internally, a *Product Graph* is implemented using the Boost Graph Library for efficiency [71].

5. *Planner* - a multi-layered motion planner that computes trajectories by searching the *Product Graph* for discrete guides.

For the co-safe LTL formulas considered in our experiments, we have converted them to minimal DFAs by using `scheck` [44]. As discussed in Section 3.1.2, φ_{cosafe} and φ_{safe} are converted into the minimum DFAs $\mathcal{A}_{\varphi_{\text{cosafe}}}$ and $\mathcal{A}_{\varphi_{\text{safe}}}$, where $\mathcal{A}_{\varphi_{\text{cosafe}}}$ accepts precisely all finite traces that satisfy φ_{cosafe}, and $\mathcal{A}_{\varphi_{\text{safe}}}$ accepts precisely all finite traces that do not violate φ_{safe}. To triangulate environments, we use Triangle [69]. All experiments were run on the Shared University Computing Grid at Rice. Each experiment used a 2.83 Ghz Intel Xeon processor with 16 GB RAM. For each set of input parameters, we average our timing measurements over 50 independent runs.

All experimental results presented in this section are meant as a proof of concept of our framework, as well as a demonstration of robustness. As was shown in previous works described in Section 2.3.2, the algorithm from which our framework extends has been shown to improve performance over monitor-based solutions (using tree-based planners without high-level guides) by orders of magnitude. It is difficult to fairly make a direct comparison between this framework and synthesis-based approaches described in Section 2.4.1, as synthesis-based approaches are assuming specific types of robotic dynamics and are not using motion planning.

4.2 Experiments

In all experiments, we use a second-order car-like robot. The robot's state is represented by $q = (x, y, \theta, v, \psi)$, which includes the planar position $(x, y) \in [0, 10]^2$, heading $\theta \in [-\pi, \pi]$, forward velocity $v \in [-1/2, 1/2]$, and steering angle $\psi \in [-\pi/6, \pi/6]$. The car is controlled with the input pair $u = (u_0, u_1)$, where $u_0 \in [-1/2, 1/2]$ is the forward acceleration and $u_1 \in [-\pi/18, \pi/18]$ is the steering angle velocity. The dynamics of the car are given by

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{l}\tan\psi \\ u_0 \\ u_1 \end{pmatrix},$$

where l is the length (the distance between the front and rear axles) of the car [47]. The car is given a sensing radius of 1. If, when executing a solution trajectory, it discovers a new obstacle within its sensing radius, the car switches to an "emergency" mode in which it applies a deceleration sufficient to reduce its velocity to $\epsilon > 0$ before colliding with the obstacle. It then patches the abstraction and the product automaton and computes a new trajectory to follow.

4.2.1 The Office-Like Environment

In this experiment, the robot is asked to visit N randomly chosen regions $p_0, \ldots, p_{N-1}$ of interest in any order, where $N \in \{1, \ldots, 5\}$, and to always avoid the sixth region p_5. Formally, the robot is given the LTL specification $\varphi_{\text{office}} = (\varphi_{\text{cosafe}}, \varphi_{\text{safe}})$, where

$$\varphi_{\text{cosafe}} = \bigwedge_{i=0}^{N-1} \mathcal{F} p_i \tag{4.1}$$

and

$$\varphi_{\text{safe}} = \mathcal{G} \neg p_5.$$

Each p_i corresponds to a propositional region in the office environment. Specifically, $p_0, \ldots, p_5$ correspond to the red, green, orange, purple, yellow, and brown regions, respectively.

The robot's initial map includes the walls of the office. However, the robot is unaware that the doors to two of the rooms are closed (we model this as rectangular obstacles filling the doorways). Figure 4.1(a) contains the actual map of the office, and Figure 4.1(b) contains the robot's initial map. We include the triangulations in the maps in Figure 4.1 to demonstrate granularity. A triangulation always respects the currently known obstacles and the geometry of the propositional regions.

Table 4.1: Experimental data for office experiment with a full initial map and a partial initial map. All times are computed in seconds and are averaged over 50 independent runs.

Initial Map	N	Solution Time (s)	Time (s) Computing and Patching Product $\mathcal{P} = \mathcal{M} \times \mathcal{A}_\varphi$
Full	1	1.08	0.005
	2	2.36	0.006
	3	5.44	0.007
	4	12.43	0.01
	5	16.44	0.012
Partial	1	2.69	0.014
	2	9.0	0.036
	3	23.08	0.102
	4	49.18	0.226
	5	80.01	0.395

Table 4.1 contains experimental data for satisfying the formula $\varphi_{\text{office}} = \left(\wedge_{i=0}^{N-1} \mathcal{F} p_i, \mathcal{G} \neg p_5\right)$ in the office environment, comparing the full initial map (Figure 4.1(a)) to a partial initial map (Figure 4.1(b)). With a fully accurate initial map, the robot does not encounter any unanticipated obstacles, and so our method behaves equivalently to the past method presented in [3–5]. We are including data for the full initial map for comparison. For the partial map, planning times increase significantly with the number of regions of interest in the coverage formula. Visiting more regions causes

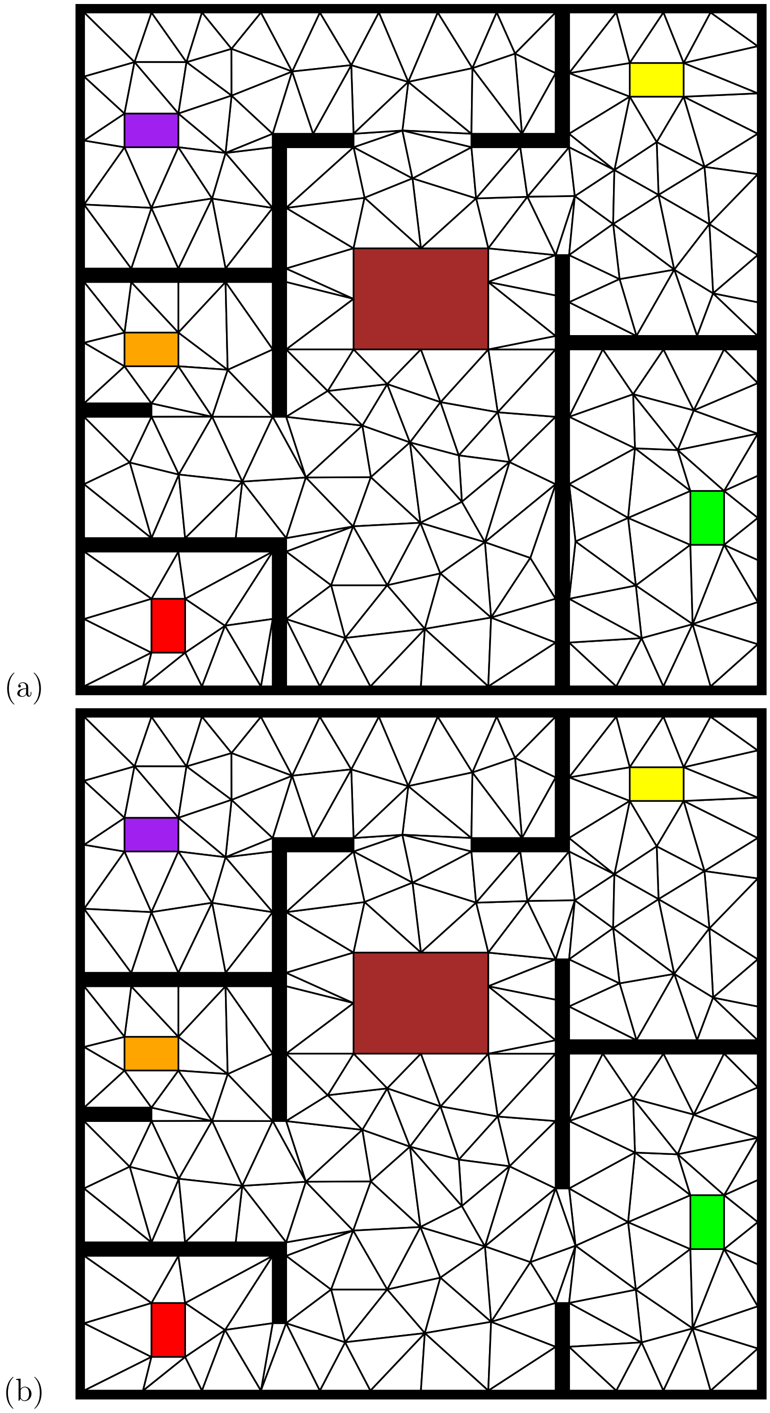


Figure 4.1: (a) an office-like environment with propositional regions of interest; (b) the robot's initial map, in which 3 obstacles are unknown.

the robot to discover more unknown obstacles, each of which requires the robot to brake. Every time the robot comes to a stop near a newly discovered obstacle, planning a solution trajectory from that stopped point is often time-consuming for the low-level motion-planning layer. This is due to the close proximity of the robot and the obstacle. With longer-range sensors, this problem can be alleviated. For all experiments, times spent computing and subsequently patching the product automaton $\mathcal{P}$ remain very small. Figure 4.2 contains an example trajectory for the robot, given a co-safe specification to visit three regions of interest (green, orange, and yellow), one of which is unreachable (green) due to a closed door, and a safe specification to avoid the brown region. This is a “raw” trajectory returned by our motion planning framework. No post-processing has been performed on the trajectory (i.e., no smoothing). First, the robot drives toward the room containing the green region. When it encounters the door, it brakes and patches the abstraction and the product automaton. The planning framework uses our measure of satisfiability to generate another trajectory that satisfies the co-safe specification as closely as possible, which is to visit the two remaining regions.

To test the importance of the initial map, we have also run experiments with initial maps of varying accuracy, ranging from a completely known environment to a completely unknown environment in which no obstacles or walls are initially known by the robot (except for the bounding box of the environment). The four types of initial maps are shown in Figure 4.3. As before, all propositional regions are initially known. Figure 4.4(a) contains average total planning times for this set of experiments. Figure 4.4(b) contains average total times spent building and patching the product

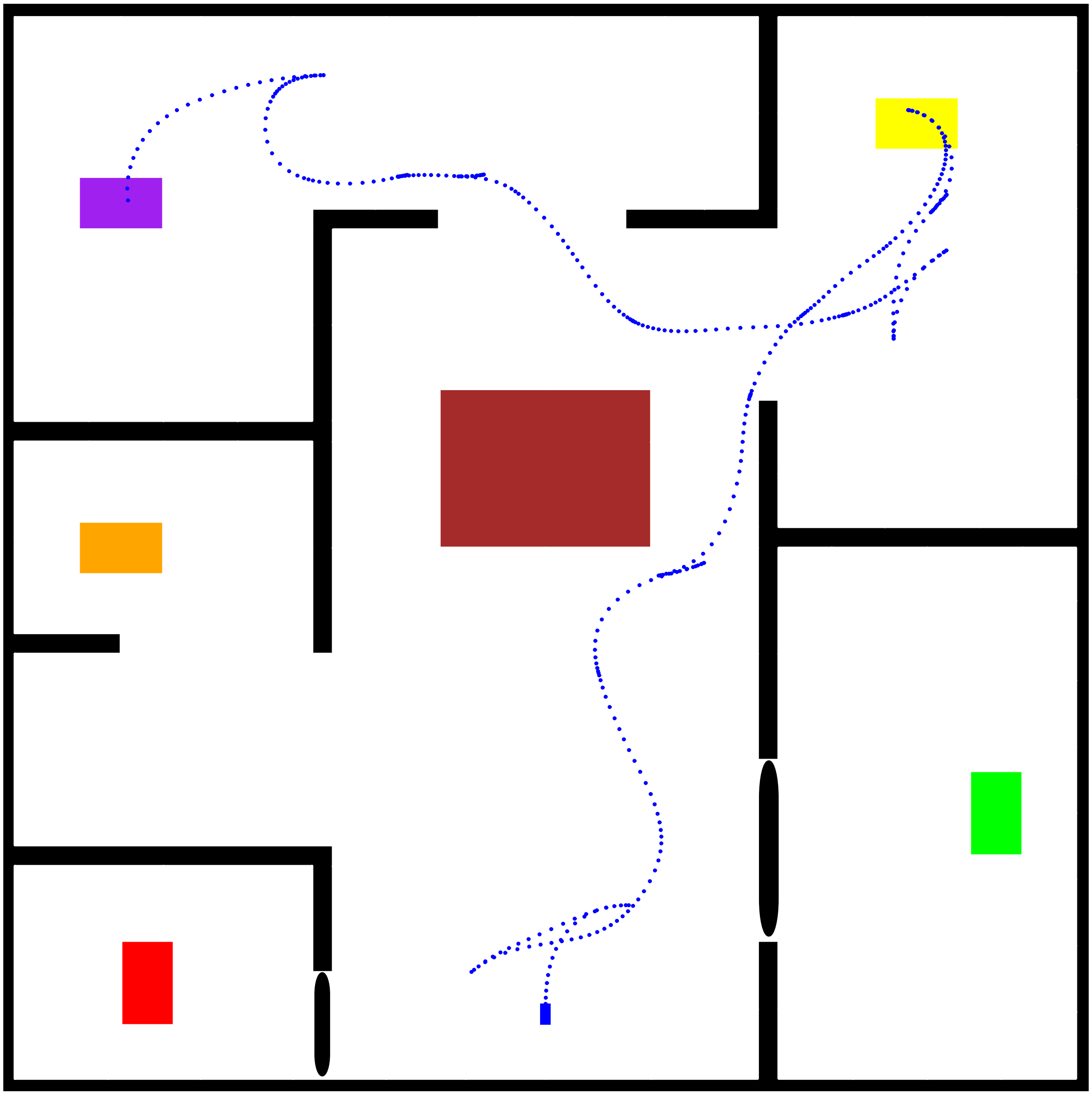


Figure 4.2: A sample trajectory that satisfies the co-safe specification "Visit the green, orange, and yellow regions in any order" as closely as possible.

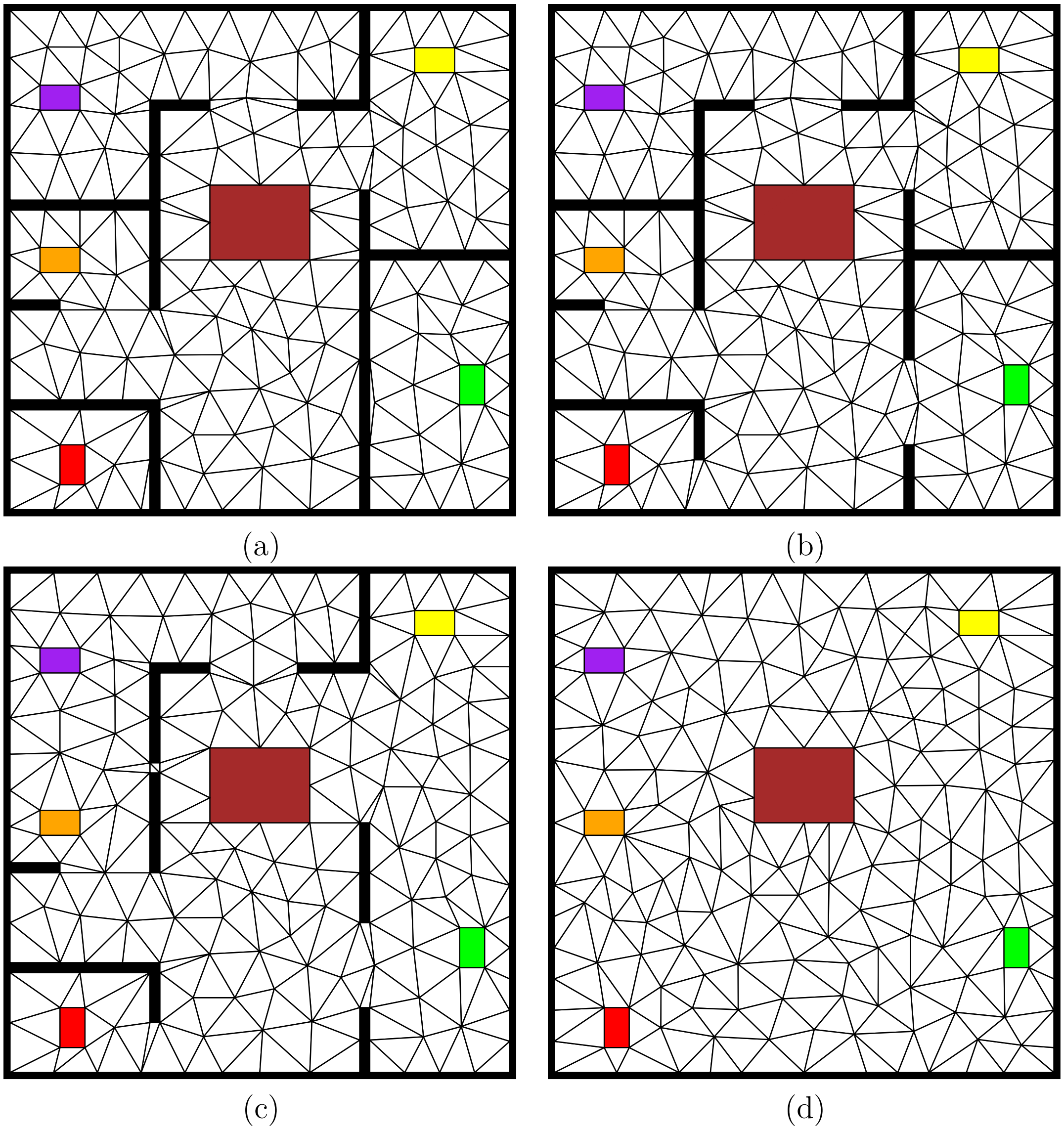


Figure 4.3: Office-like environments in which (a) all obstacles are known; (b) 3 obstacles are unknown; (c) 6 obstacles are unknown; (d) all 16 obstacles are unknown.

automaton. In these experiments, we focus on solving φ_5, the most difficult of the formulas to consider. The time spent building and patching the product automaton is negligible compared to the time spent planning solution trajectories. Moreover, all solution times scale with the number of unknown obstacles that are discovered in the environment.

In Section 2.1, we described our framework's ability to efficiently deal with a large number of potential environmental unknowns as an advantage over synthesis-based approaches which become intractable with many unknowns. The timing data in Figure 4.4(a) reveals a weakness in our framework, specifically that the planning time scales with the number of discovered obstacles. Based on this data, we conclude that this framework is most effective when the robot's initial map of the environment is mostly accurate. The robot must recompute a solution trajectory for every obstacle that is discovered, and with a completely unknown environment (the rightmost bar in which 15 obstacles are unknown), the wasted planning time can add up.

4.2.2 The Maze-Like Environment

In this experiment, the robot is given a more complex goal. The maze-like environment contains six propositional regions, p_0 through p_5, corresponding to the red, green, orange, purple, yellow, and brown regions in Figure 4.5.

The robot is given the LTL specification $\varphi_{\text{maze}} = (\varphi_{\text{cosafe}}, \varphi_{\text{safe}})$, where

$$\varphi_{\text{cosafe}} = \mathcal{F} p_0 \wedge \mathcal{F} p_5,$$

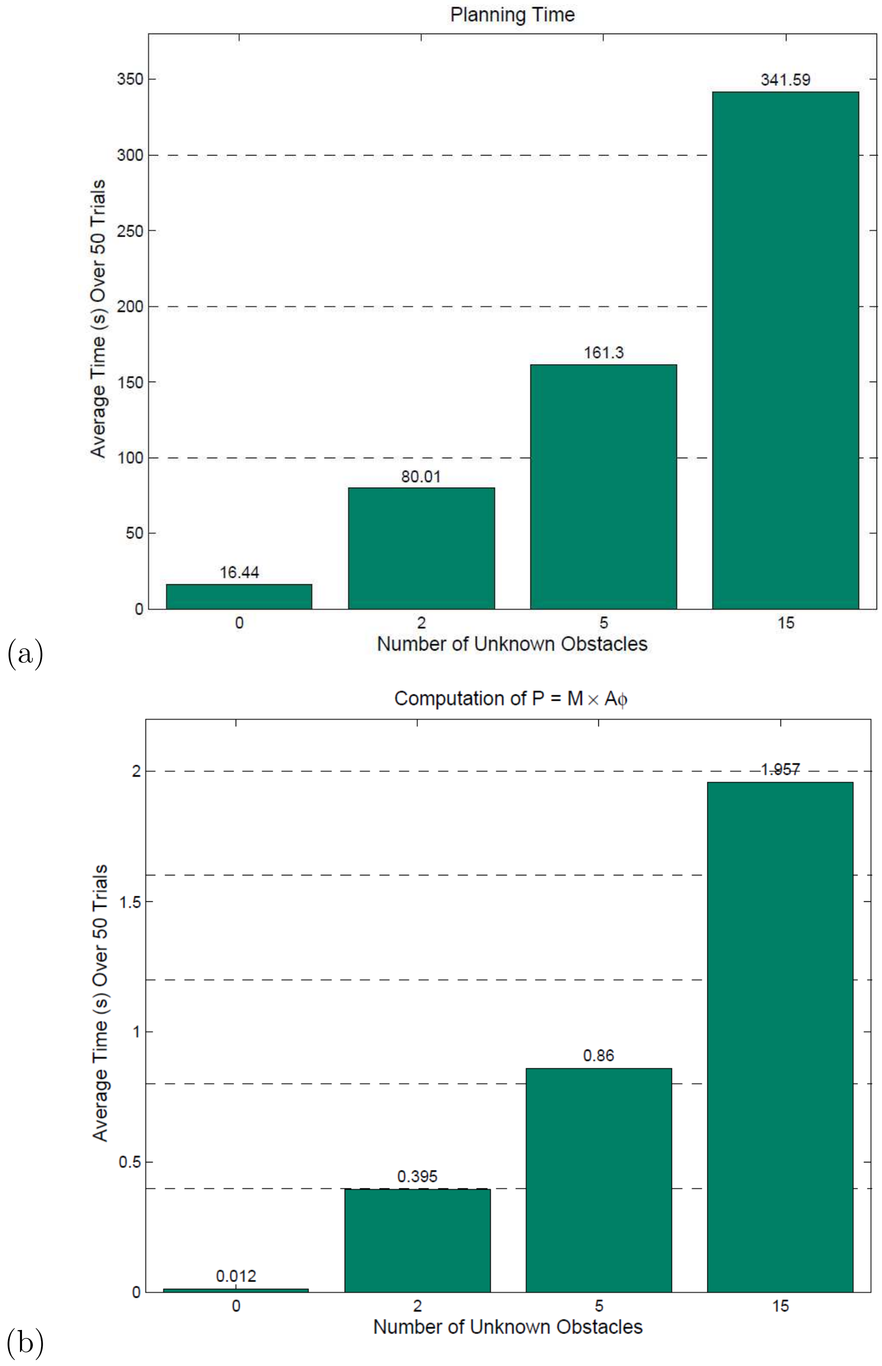


Figure 4.4: Average total time spent (a) planning and (b) building and patching the product automaton with the office environment using formula φ_5 and varied initial maps. All times are computed in seconds and are averaged over 50 independent runs.

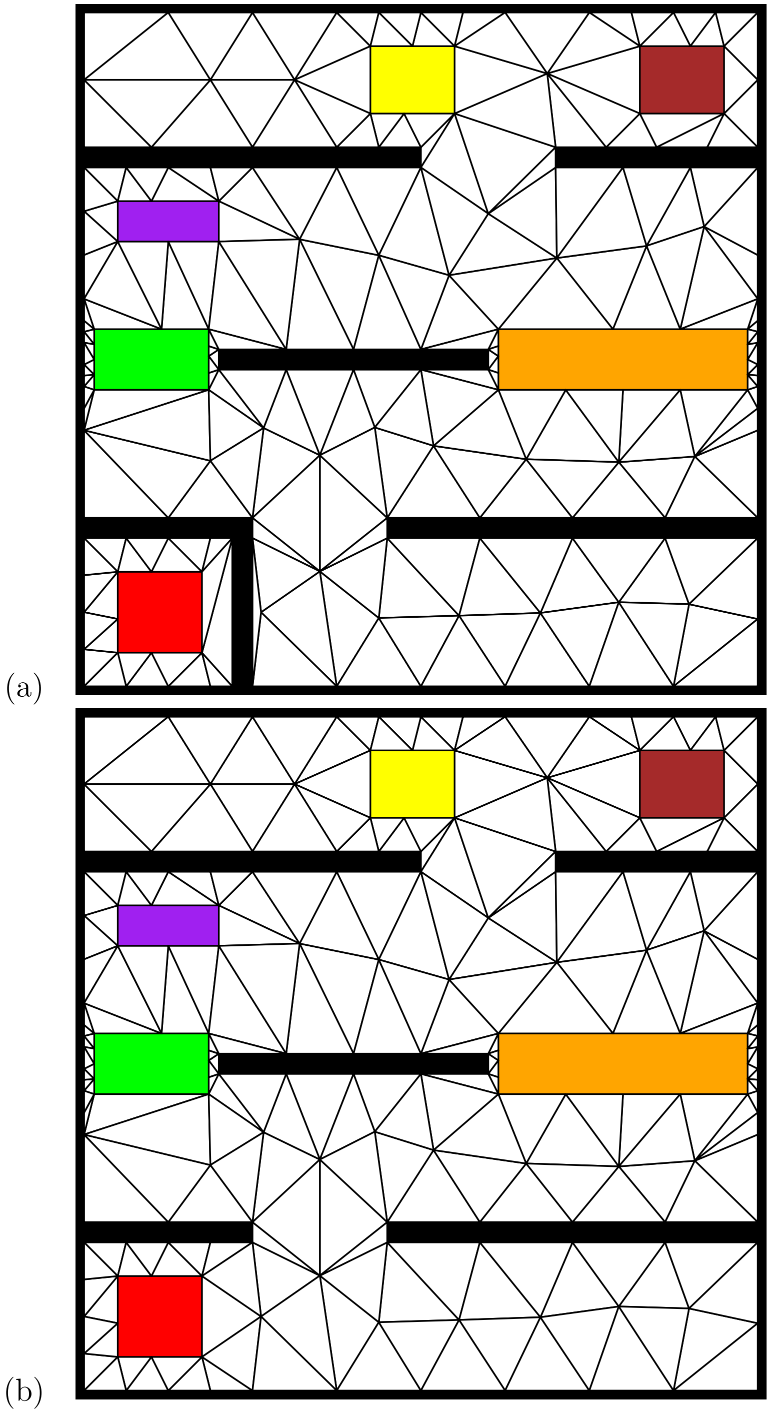


Figure 4.5: (a) a maze-like environment with propositional regions of interest; (b) the robot's initial map of the maze, in which one obstacle is unknown.

which specifies that the robot should visit the red and brown regions in any order, and

$$\varphi_{\text{safe}} = \mathcal{G}\neg p_3 \wedge \mathcal{G}\Big(p_2 \to \mathcal{G}\neg p_4\Big),$$

which specifies that if the robot should always avoid p_3, and if it ever visits p_2, then it should subsequently always avoid p_4. With φ_{safe}, we are giving the robot two options on how to deal with the "fork in the road" in the middle of the maze environment. If the robot takes the left path and drives through p_1, the green region, then it must take special care to avoid visiting p_3, the purple region which is nearby. If, on the other hand, the robot takes the right path and drives through p_2, the orange region, then the robot cannot touch p_4 (the yellow region) on its way to p_5. The DFA $\mathcal{A}_{\varphi_{\text{cosafe}}}$, which accepts precisely all finite traces that satisfy φ_{cosafe}, is given in Figure 4.6. The DFA $\mathcal{A}_{\varphi_{\text{safe}}}$, which accepts precisely all finite traces that do not violate φ_{safe}, is given in Figure 4.7. Figure 4.7 also illustrates how we obtain $\mathcal{A}_{\varphi_{\text{safe}}}$: by computing the DFA corresponding to $\neg\varphi_{\text{safe}}$, flipping its acceptance condition, and minimizing.

The robot's initial map of the maze, given in Figure 4.5(b), includes all obstacles except for the wall blocking the red region (p_0). Figure 4.5(a) contains the actual map of the maze.

Table 4.2 contains experimental data for satisfying the formula φ_{maze} in the maze environment, comparing the full initial map (Figure 4.5(a) to the partial initial map Figure 4.5(b). As with the office environment, with a fully accurate initial map, our method behaves equivalently to `ML-LTL-MP` [3–5]. The maze environment is an example in which our framework does quite well, as the robot's initial map of the

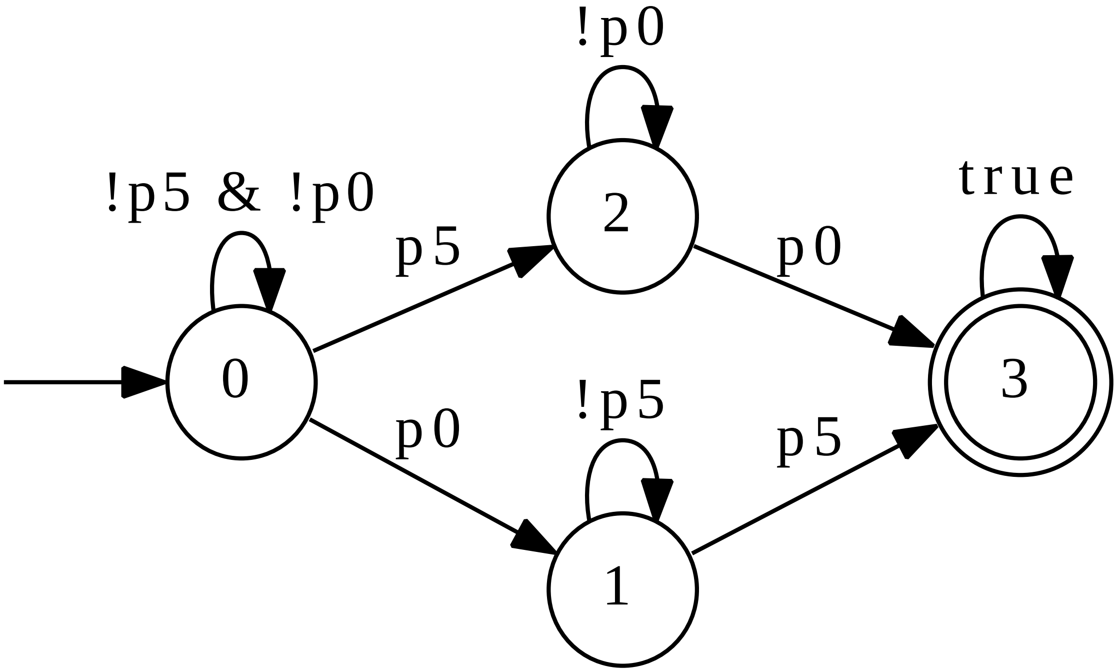


Figure 4.6: The minimal DFA $\mathcal{A}_{\varphi_{\text{safe}}}$, corresponding to the safe component φ_{cosafe} of the LTL specification φ_{maze}.

environment is almost completely accurate. Beginning with a partial initial map, once the robot discovers the wall blocking the red region, it must brake and recompute one more trajectory to execute, finishing with a solution time approximately twice that of when the robot has a full initial map.

Table 4.2: Experimental data for the maze experiment with a full initial map and a partial initial map. All times are computed in seconds and are averaged over 50 independent runs.

Initial Map	Solution Time (s)	Time (s) Computing and Patching Product $\mathcal{P} = \mathcal{M} \times \mathcal{A}_\varphi$
Full	9.54	0.008
Partial	23.1	0.062

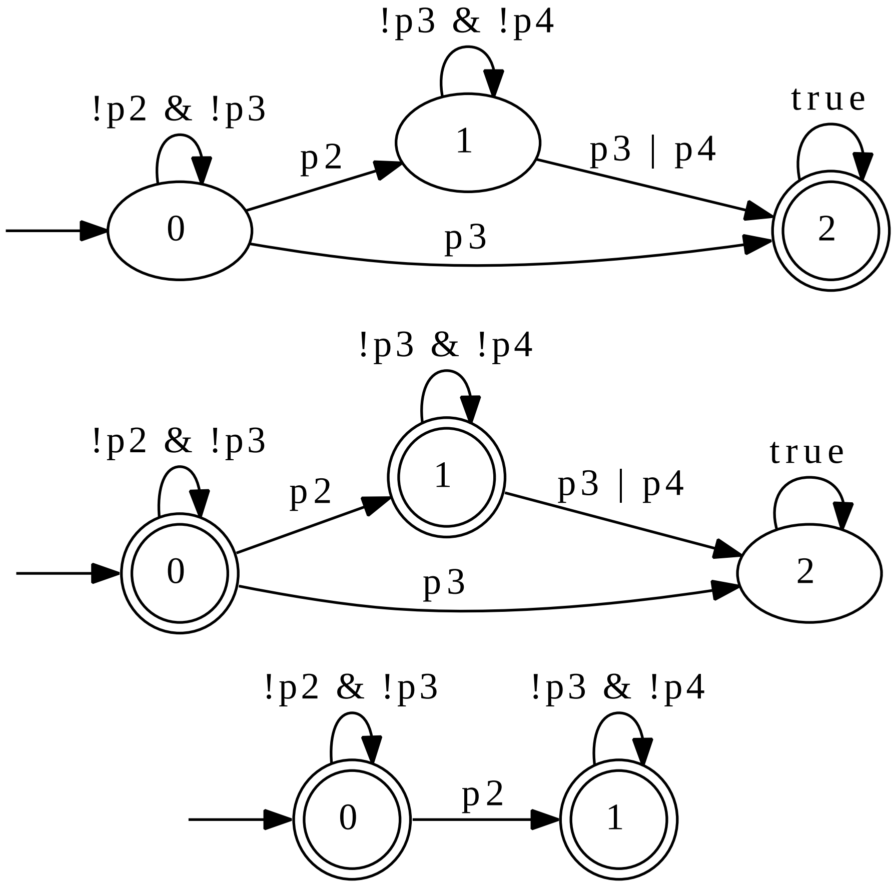


Figure 4.7: The conversion from the DFA corresponding to $\neg\varphi_{\text{safe}}$ to the minimal DFA $\mathcal{A}_{\varphi_{\text{safe}}}$.

Chapter 5

Possible Extensions

In this chapter we discuss one immediate possible extension to our framework, related to our use of logical specifications. As we discuss in the following section, the product automaton used by our framework is general enough to easily be extended to multiple types of temporal logic formulas.

5.1 Hard and Soft Constraints

The formula φ_{safe}, as we have used it in our framework, does not actually have to be a safety formula. Instead, it can just be the components of the temporal logic specification that we wish to treat as a "hard" constraint. Similarly, $\varphi_{\mathrm{cosafe}}$, as we have used it, just needs to encode the "soft" constraints of the specification.

The issue of hard versus soft constraints is a very rich area of research. For one, in the area of classical AI planning, a *constraint satisfaction problem* (CSP) is a graph-based search problem with a set of rigid (hard) constraints [67]. *Flexible CSPs* are an extension of CSPs in which a solution is allowed to violate some (soft) constraints [20, 24, 55]. Typically, solutions to flexible CSPs are ranked according to how many soft constraints they violate, and each constraint can be assigned a weight value to control its priority of satisfaction. Additionally, *preference-based planning* is a type of classical planning in which all constraints are soft [8, 18, 25, 72]. Typically, as with flexible CSPs, paths in preference-based planning are ranked according to the number of preferences they satisfy. The Planning Domain Definition Language (PDDL) is an example of an AI planning language that supports preference-based planning [26, 54].

In the case of our framework, we imagine a simple extension of our framework that accepts as input an arbitrary number of hard and soft constraint LTL formulas. We assume that the hard constraint formulas are ones that cannot be violated by the robot, and the soft constraint formulas are ones that we allow to violate if necessary, but will satisfy as closely as possible. The product automaton through which we guide a low-level planner would then be computed as

$$\mathcal{P} = \mathcal{M} \times \prod_{\mathcal{H} \in \textsc{Hard}} \mathcal{H} \times \prod_{\mathcal{S} \in \textsc{Soft}} \mathcal{S}.$$

Assuming $\textsc{Hard} = \{\mathcal{H}_1, \ldots, \mathcal{H}_k\}$ and $\textsc{Soft} = \{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$, where each $\mathcal{H}_i \in \textsc{Hard}$ is a DFA encoding a hard constraint formula, and each $\mathcal{S}_j \in \textsc{Soft}$ is a DFA encoding a soft constraint formula, a lead through the product automaton $\mathcal{P}$ would then be a sequence of high-level states

$$\left(\left(d_0, \{h_0^i\}_{i=1}^k, \{s_0^j\}_{j=1}^l \right), \ldots, \left(d_g, \{h_g^i\}_{i=1}^k, \{s_g^j\}_{j=1}^l \right) \right)$$

such that h_g^i is accepting in $\mathcal{H}_i$ for each i, and $\textsc{DistFromAcc}(s_g^j, \mathcal{S}_j)$ is minimized for each j.

5.2 Beyond Co-safe and Safe LTL

It is important to note that because our framework accepts arbitrary DFAs as inputs to represent specifications, we can support more than just co-safety and safety formulas. To illustrate this, we adapt the maze experiment from Section 4.2.2; we move the yellow region of interest to the top-left corner of the map. As before, the robot's initial map does not contain the wall blocking the red region. Figures 5.1(a) and 5.1(b) contain the actual map of the maze environment and the robot's initial map, respectively.

We generalize the specifications given to the robot in terms of "soft" and "hard" constraints as described in the previous section. As before, the soft constraint specification component is for the robot to "visit the red and brown regions in any order", given by the formula

$$\varphi = \mathcal{F} p_0 \wedge \mathcal{F} p_5.$$

We change the hard constraint specification to be the following command: "Always avoid the purple region. Each time you visit the orange region, then you should

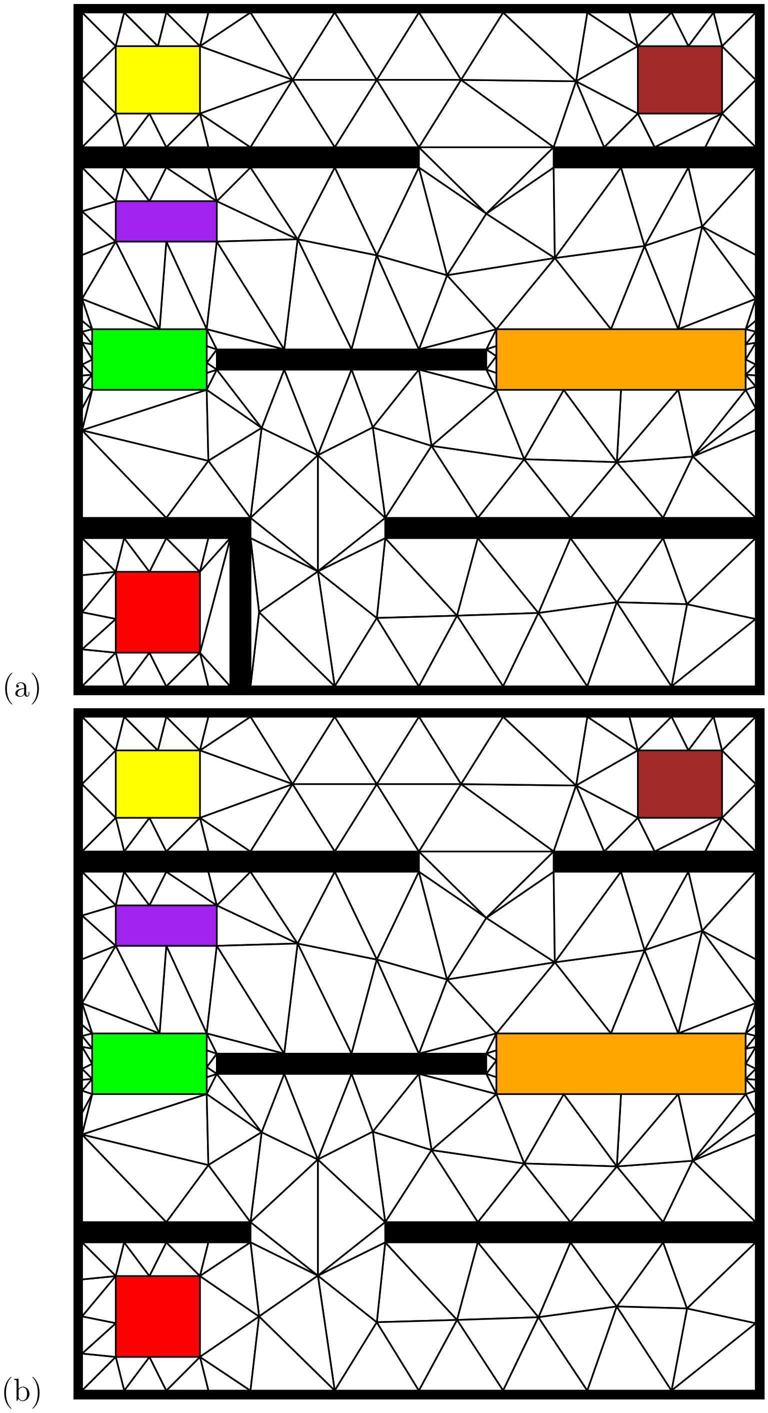


Figure 5.1: (a) A maze-like environment with propositional regions of interest; (b) the robot's initial map of the maze, in which one obstacle is unknown.

immediately visit the yellow region afterwards." This command is given by the formula

$$\psi = \mathcal{G}(\neg p_3) \wedge \mathcal{G}\Big(p_2 \rightarrow \neg(p_0 \vee p_1 \vee p_3 \vee p_4 \vee p_5)\mathcal{U}p_4)\Big).$$

Notice that ψ is neither a co-safety formula nor a safety formula. For one, any finite trace that satisfies ψ can be extended with behavior that visits the purple region, which sets p_3 to true and violates ψ. Additionally, there exist finite traces that violate the right conjunct of ψ (such as visiting orange region and staying there) which can be extended with behavior to satisfy ψ (such as leaving the orange region and then immediately visiting the yellow region). The formula ψ is a *liveness property*. Even though ψ is neither a co-safety or safety formula, we can still manually construct a minimal DFA that accepts precisely all finite traces that satisfy ψ (here we loosely use the term *satisfy* to mean *satisfies over finite semantics*, i.e., ends in an accepting state). Figure 5.2 contains such a DFA. Our framework works well with such a DFA. Table 5.1 contains experimental data for the robot satisfying the specification (φ, ψ) in the maze-like environment.

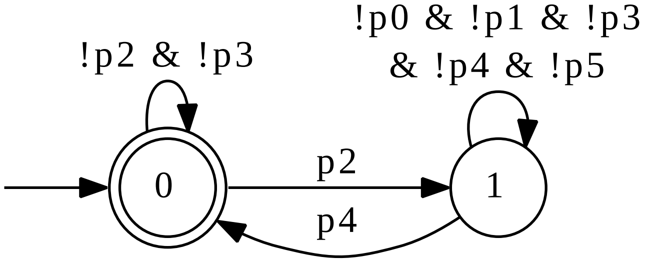


Figure 5.2: The minimal DFA $\mathcal{A}_\psi$, corresponding to the liveness formula ψ.

Table 5.1: Experimental data for maze with a partial initial map.

Solution Time	Time Computing Product $\mathcal{P} = \mathcal{M} \times \mathcal{A}_\varphi$
18.34	0.3

Chapter 6

Conclusion and Future Work

In this thesis, we have presented an iterative motion planning framework for a robotic system with complex and possibly nonlinear dynamics given a partially unknown environment and a temporal logic specification consisting of co-safe and safe formula components. We have also presented a measure of satisfiability which we can optimize in cases where obstacles in the environment prevent full satisfaction of the co-safe component of the given temporal logic specification. In such cases, the robotic system satisfies the co-safe specification as closely as possible, while still guaranteeing that the robotic system does not violate the safe specification.

For future work, we plan to add support for obstacles to disappear from the robot's initial map (the current framework only supports obstacles appearing). We could also assume a probabilistic distribution on where and when obstacles will appear, and then generate trajectories that maximize probability of successful satisfaction of the specification. In addition, we would like to consider a "greedy" temporal motion planning approach that begins executing a partial trajectory along a lead in the product automaton. This is to prevent the framework from wasting time generating an entire solution trajectory for a large specification, only to discover an obstacle early in that trajectory, stop, and recompute another solution trajectory. As seen in our experiments, when the robot's initial map is sufficiently inaccurate, this wasted planning time can add up.

One strength of our framework lies in our ability to consider any type of polygonal obstacle being discovered in any part of the environment. With alternative synthesis-based approaches, to consider such a large number of unknowns would be intractable. However, as we discussed in our experimental results in Chapter 4, our solution times scale with the number of obstacles that are discovered by the robot, and therefore our framework will perform best when only a few obstacles are missing from the robot's initial map of the environment. Additionally, our approach is novel in how we deal with a newly discovered obstacle. Because the product automaton, the high-level structure we use for planning, keeps the task specification and the workspace abstraction separate, reworking the product automaton to incorporate the new obstacle does not require recomputing the automata corresponding to the specifications, which is the most time-intensive task.

At the same time, our framework should not be viewed as completely separate from the synthesis-based approaches discussed in Section 2.4.1. The two approaches can certainly be connected. For one, a robot with known dynamics and a bisimilar abstraction of the environment can certainly be used in our framework. The high-level layer, where the product automaton consists of a product of the workspace abstraction and the automata corresponding to the task specifications, would remain exactly the same. The low-level layer would be significantly simpler. Because of the bisimilarity property between the robot's dynamics and the discrete workspace abstraction, any high-level guide computed through the product automaton would be realizable by the robot. So, the first high-level guide computed by our framework could be sent to the robot, and the corresponding controllers could be generated to take the robot through each region in the guide. When a new obstacle is discovered and the abstraction is recomputed, it is likely that the bisimilarity property can no longer be preserved. In such cases, the framework could "fall back" on motion planning to complete the task.

It is clear that robotic planning with temporal logic specifications remains a fertile area for research. Referring back to the continuum in Figure 1.1 in Chapter 1, planning for tasks such as "cook dinner" requires strong notions of logical and dynamical constraints, as well as support for dealing with uncertainties both internally and externally. This thesis touches upon several of these issues, specifically how to satisfy logical constraints in light of external uncertainties. Together with the related work being done on these and the other issues in constraints and uncertainties, many of which we have discussed in Chapter 2, we as a research community are incrementally moving along the planning continuum from Figure 1.1. As robots become increasingly ubiquitous, the research done in this area will prove to be one of the many key foundations in designing autonomous robots to work in the presence of humans.

Bibliography

[1] Kostas E. Bekris, Devin K. Grady, Mark Moll, and Lydia E. Kavraki. Safe distributed motion coordination for second-order systems with different planning cycles. *Intl. J. of Robotics Research*, 31(2):129–149, February 2012.

[2] Kostas E. Bekris and Lydia E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *IEEE Intl. Conf. on Robotics and Automation*, pages 704–710, 2007.

[3] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Motion planning with hybrid dynamics and temporal goals. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 1108 –1115, Dec. 2010.

[4] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2689 –2696, May 2010.

[5] A. Bhatia, M.R. Maly, L.E. Kavraki, and M.Y. Vardi. Motion planning with complex goals. *Robotics Automation Magazine, IEEE*, 18(3):55 –64, Sep. 2011.

[6] R. Bloem, K. Greimel, T.A. Henzinger, and B. Jobstmann. Synthesizing robust systems. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 85–92. IEEE, 2009.

[7] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv SaÊijar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[8] Ronen I Brafman and Yuri Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of ICAPS*, pages 182–191, 2005.

[9] B. Burns and O. Brock. Single-query motion planning with utility-guided random trees. In *IEEE Intl. Conf. on Robotics and Automation*, pages 3307–3312, April 2007.

[10] John Canny. Some algebraic and geometric computations in PSPACE. In *Annual ACM Symposium on Theory of Computing*, pages 460–469, Chicago, Illinois, United States, 1988. ACM Press.

[11] Yushan Chen, J. Tumova, and C. Belta. LTL robot motion control based on automata learning of environmental dynamics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5177 –5182, May 2012.

[12] Peng Cheng, George Pappas, and Vijay Kumar. Decidability of motion planning with differential constraints. In *IEEE Intl. Conf. on Robotics and Automation*, pages 1826–1831, 2007.

[13] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, 2005.

[14] Ioan A. Şucan. *Task and Motion Planning for Mobile Manipulators.* PhD thesis, Rice University, Department of Computer Science, Houston, TX, August 2011.

[15] Ioan A. Şucan and Lydia E. Kavraki. On the implementation of single-query sampling-based motion planners. In *IEEE Intl. Conf. on Robotics and Automation*, pages 2005–2011, may 2010.

[16] Ioan A. Şucan and Lydia E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Trans. on Robotics*, 28(1):116–131, 2012.

[17] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 2012. Accepted for publication.

[18] James P Delgrande, Torsten Schaub, and Hans Tompits. A general framework for expressing preferences in causal reasoning and planning. *Journal of Logic and Computation*, 17(5):871–907, 2007.

[19] X.C. Ding, S.L. Smith, C. Belta, and D. Rus. MDP optimal control under temporal logic constraints. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 532–538. IEEE, 2011.

[20] Didier Dubois, Helene Fargier, and Henri Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6(4):287–309, 1996.

[21] G. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45:343–352, 2009.

[22] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 1988 – 1993, 2010.

[23] T. Fraichard. A short paper about motion safety. In *Proc. 2007 IEEE Intl. Conf. on Robotics and Automation*, pages 1140–1145, April 2007.

[24] Eugene C Freuder, Richard J Wallace, and Robert Heffernan. Partial constraint satisfaction. In *Artificial Intelligence*, 1992.

[25] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. *The Language of the Fifth International Planning Competition. Tech. Rep. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 2005.

[26] Alfonso Gerevini and Derek Long. Preferences and soft constraints in pddl3. In *ICAPS workshop on planning with preferences and soft constraints*, pages 46–53, 2006.

[27] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[28] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Intl. J. of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

[29] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic planning with moving obstacles. *Intl. J. of Robotics Research*, 21(3):233–255, March 2002.

[30] Karaman and Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control (CDC)*, Shanghai, China, Dec. 2009.

[31] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, August 1996.

[32] Kangjin Kim, G.E. Fainekos, and S. Sankaranarayanan. On the revision problem of specification automata. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5171 –5176, may 2012.

[33] Kangjin Kim and Georgios Fainekos. Approximate solutions for the minimal revision problem of specification automata. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 265–271, 2012.

[34] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. From structured english to robot motion. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2717 –2722, 29 2007-nov. 2 2007.

[35] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Where's waldo? Sensor-based temporal logic motion planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116 –3121, Apr. 2007.

[36] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25(6):1370–1381, Dec. 2009.

[37] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive, high-level robot control. *Robotics Automation Magazine, IEEE*, 18(3):65 –74, Sep. 2011.

[38] James Kuffner and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proc. 2000 IEEE Intl. Conf. on Robotics and Automation*, pages 995–1001, San Francisco, CA, April 2000.

[39] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199 – 213. Springer-Verlag, 2007.

[40] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19:291 – 314, 2001.

[41] A. M. Ladd and L. E. Kavraki. Fast tree-based exploration of state space for robots with dynamics. In M. Erdmann, D. Hsu, M. Overmars, and A. F.

van der Stappen, editors, *Algorithmic Foundations of Robotics VI*, pages 297–312. Springer, STAR 17, 2005.

[42] M. Lahijanian, J. Wasniewski, S.B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees,. In *IEEE International Conference on Robotics and Automation*, pages 3227–3232, Anchorage, Alaska, 2010.

[43] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, Apr. 2012.

[44] T. Latvala. Efficient model checking of safety properties. In *Model Checking Software*, pages 74–88. Springer, 2003.

[45] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, October 1998.

[46] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Intl. J. of Robotics Research*, 20(5):378–400, May 2001.

[47] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[48] Scott C. Livingston and Richard M. Murray. Just-in-time synthesis for motion planning with temporal logic. In *IEEE Intl. Conf. on Robotics and Automation*, 2013. To appear.

[49] Scott C. Livingston, Richard M. Murray, and Joel W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *ICRA*, pages 5163–5170, 2012.

[50] Scott C. Livingston, Pavithra Prabhakar, Alex B. Jose, and Richard M. Murray. Patching task-level robot controllers based on a local μ-calculus formula. In *IEEE Intl. Conf. on Robotics and Automation*, 2013. To appear.

[51] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Computing*, 32(2):108–120, 1983.

[52] Matthew R. Maly, Morteza Lahijanian, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in

partially unknown environments. In *Proceedings of the 16th ACM International Conference on Hybrid Systems: Computation and Control*, 2013. To appear.

[53] M.R. Maly and L.E. Kavraki. Low-dimensional projections for syclop. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 420 –425, oct. 2012.

[54] Drew M McDermott. The 1998 ai planning systems competition. *AI magazine*, 21(2):35, 2000.

[55] Ian Miguel. *Dynamic flexible constraint satisfaction and its application to AI planning*. Springer-Verlag New York Incorporated, 2004.

[56] J. M. Phillips, N. Bedrosian, and L. E. Kavraki. Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces. In *Proc. 2004 IEEE Intl. Conf. on Robotics and Automation*, pages 3968–3973, New Orleans, LA, April 2004. IEEE Press.

[57] Nir Piterman, Amir Pnueli, and Yaniv SaâĂŹar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[58] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Discrete search leading continuous exploration for kinodynamic motion planning. In *Robotics: Science and Systems*, Atlanta, Georgia, 2007.

[59] E. Plaku, L.E. Kavraki, and M.Y. Vardi. Impact of workspace decompositions on discrete search leading continuous exploration (dslx) motion planning. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3751 –3756, may 2008.

[60] E. Plaku, L.E. Kavraki, and M.Y. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Trans. on Robotics*, 26(3):469–482, Jun. 2010.

[61] E. Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Falsification of LTL safety properties in hybrid systems. In *Proc. of the Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, York, UK, 2009.

[62] E. Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*, 34:157–182, 2009.

[63] Erion Plaku, Lydia Kavraki, and Moshe Vardi. Falsification of ltl safety properties in hybrid systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 2012.

[64] Vasumathi Raman and Hadas Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP. In *Proceedings of the 23rd International Conference on Computer Cided Verification*, CAV'11, pages 663–668, Berlin, Heidelberg, 2011. Springer-Verlag.

[65] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. Foundations of Computer Science*, pages 421–427, 1979.

[66] S. Rodriguez, Xinyu Tang, Jyh-Ming Lien, and N.M. Amato. An obstacle-based rapidly-exploring random tree. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 895 –900, may 2006.

[67] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, 1995.

[68] Shahar Sarid, Bingxin Xu, and Hadas Kress-Gazit. Guaranteeing high-level behaviors while exploring partially known maps. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[69] J. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, chapter 23, pages 203 – 222. Springer-Verlag, Berlin/Heidelberg, 1996.

[70] A. Shkolnik, M. Walter, and R. Tedrake. Reachability-guided sampling for planning under differential constraints. In *IEEE Intl. Conf. on Robotics and Automation*, pages 2859–2865, 12-17 2009.

[71] Jeremy G. Siek, Lee-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual.* Addison-Wesley, 2002.

[72] Menkes Van Den Briel, Romeo Sanchez, Minh B Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 562–569. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

[73] Pavol Černý, Sivakanth Gopi, Thomas A. Henzinger, Arjun Radhakrishna, and Nishant Totla. Synthesis from incompatible specifications. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '12, pages 53–62, New York, NY, USA, 2012. ACM.

[74] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 101–110. ACM, 2010.

[75] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding horizon temporal logic planning. *Automatic Control, IEEE Transactions on*, 57(11):2817 –2830, nov. 2012.