GUARANTEEING HIGH-LEVEL ROBOT BEHAVIOURS WITH PAST MEMORY AND FUTURE UNKNOWN

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Bingxin Xu

January 2013

ABSTRACT

This thesis presents the development of high-level guaranteed robot control for reactive behaviours by relaxing two assumptions: 1) the workspace is well-known in advance; 2) the temporal logic based formalisms encode the specification by expressing the properties of future paths. This paper addresses the challenges of relaxing both of the assumptions by presenting an approach for automatically re-synthesizing a hybrid controller that guarantees user-defined high-level robot behaviour while exploring and updating a partially unknown workspace, and providing a concise grammar for specifying high-level tasks that require memory of past events.

In the first challenge, this thesis introduces an approach that includes dynamically adding new regions into the workspace during execution, automatically rewriting the specification, and re-synthesizing the controller while preserving the robot state and its history of task completion. The approach is implemented within the LTLMoP toolkit and is demonstrated in an experiment.

For the second challenge, this thesis introduces an innovative structured English grammar for specifying high-level behaviour that automatically performs memory operations without requiring explicit definition from the specification designer. This grammar admits intuitive, unambiguous specifications for tasks that implicitly use memory for purposes including non-repeated goals, strictly ordered requirements, etc. The approach is also implemented within the LTLMoP toolkit.

BIOGRAPHICAL SKETCH

Bingxin Xu accepted the offer of joining this world at the end of January in 1988, somewhere in Shanghai, China. He didn't remember the rest of the details because he was so distracted by the exciting and amazing adventure lying ahead of him.

22 years later, Bingxin graduated from University of Michigan, Ann Arbor in Ann Arbor, Michigan. He accepted another important offer of his life from Cornell University. Since then he spent two years in the Department of Mechanical Engineering as a student and researcher. Two years are absolutely too short to enjoy his life in Cornell. Now he is approaching the end of this short-term journey, and he is extremely excited to wrap up what he has learned in the past two years.

This document is dedicated to all Cornell graduate students.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Hadas Kress-Gazit for the continuous support of my M.S. study and research, for her patience, motivation, enthusiasm, immense knowledge, and above of all, enlightening me the first glance of research. Her guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the other thesis committee: Prof. Brandon Hencey, for his encouragement, comments, and hard questions.

I thank my fellow group mates in Autonomous Systems Laboratory: Jon DeCastro, Cameron Finucane, Jim Jing, Ben Johnson, Vasu Raman, Shahar Sarid, for the stimulating discussions, for the hard working before deadlines, and for all the fun we have had in the last few years.

Last but not least I would like to give my special thanks to my family: my parents Zumou Xu and Wei Yu, for opening the door of this world at the first place, and supporting me practically and spiritually throughout my life; and also my life partner and soul mate, Tian Zhang, for being with me in whatever situations.

To them I dedicate this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Robotics is the integration of computers and controlled mechanisms to make devices reprogrammable and versatile. Different robots are used in most industries and will be used even more in the decade to come. The use of robots in the U.S. is growing rapidly.

As the original definition of itself, robotics addresses the automation of mechanical systems that have multiple parts with versatile functions, including sensing, locomotion, actuation, etc. And the continuous addition of new devices and new functions are also expected. Being a specialist in Robotics is more like being a generalist in a technical way, because we are dealing with a large amount of components. We have different robots, we want the robots to work in different dynamic environments, and fulfil different tasks. Each type of the robots is like a totally different system, which takes significant time and labor if every single task is programmed separately and independently.

One of the most important needs is to design a universal framework that allows users to easily integrate different components together and use them in a relatively arbitrary combination. In addition, since it is unreasonable to require the users to specify the details of controlling every single components, the following need with almost equal priority is to convert high-level specifications of tasks from human input into low-level descriptions of how to control the local parts.

Prof. Hadas Kress-Gazit and her research group developed such a framework [11] that answered both of the needs. The framework is able to automatically translate high-level human input into logic formulas, synthesize to get a guaranteed correct hybrid controller, and execute the controller to perform the correct, reactive behavior. The approach tackles the continuous problem of robot sensing, motion and action by creating

a discrete abstraction of the task, generating a provably-correct discrete solution and implementing the solution by composing a set of continuous low-level controllers such that the overall continuous behavior of the robot is the desired one. The specification formalisms is based on Linear Temporal Logic (LTL). The creation of the discrete solution is based on ideas from model checking and synthesis when assuming no forms of uncertainty.

There are two significant assumptions in the original approach: 1) The workspace is assumed to be well-defined in advance, which means all the discrete models are complete and predictable. It is because the fundamental concept of the discrete planning algorithms is strictly based on the finite state space. This assumption limits the capability of robot behaviors in scenarios such as exploring and mapping tasks. 2) The LTL formula only expresses the properties of current and future states in the path. The robot is able to perform the correct reaction to the environment input and achieve the goal requirements, while what happened in the past is forgotten. There are scenarios, like non-repeating goals, strictly ordered tasks, that the memory of the past events is extremely important.

This thesis aims to develop the control framework by relaxing the two assumptions without losing correctness and safety of the original system.

1.2 Related Work

This work builds on the framework introduced in [11]. The framework uses formal methods to automatically generate controllers to achieve the high-level tasks specified by the users. A complete process of generating a guaranteed correct robot controller includes several steps described as followed.

First, users are allowed to use structured English to specify the high-level robot tasks. There are several works that use language for robots control, including mapping high-

level instructions to sequences of commands in an external environment [4], translating natural language directives into temporal and dynamic logic representation [7], figuring out directions using statistical machine translation [13], etc. The input language used in this work is template-based structured English instead of natural language, which is easy to learn and avoids the uncertainty and ambiguity of natural language.

Second, the structured English is translated into the linear Temporal Logic (LTL)[15] formulas to express the specifications. The purpose of using LTL is to apply formal methods such as model-checking [5] and synthesis methods [11, 14, 19] to plan the robot motion. The given map of the obstacle-free workspace is decomposed into cells, and the adjacency relationship between the cells is determined and represented by a discrete graph [2] with nodes representing regions and edges indicating the transition relationship. The discrete path that agrees with the specification is generated by a search in the discrete space[14].

Finally, the discrete solution is implemented by low-level controllers that execute the local continuous motion and actions. A discrete trajectory over the sequence of cells is generated, and the local controllers are constructed to follow the trajectory. There are multiple choices of the local controllers, like potential field type planners [10] and sampling-based approaches [3, 9]. A convex partition of the polygonal map is considered since there are existing controllers that are guaranteed to drive a robot from any point within a convex polygon to anyone of its faces, and thus to the adjacent region (e.g. [1, 6, 12]). It assumes no noise on the local sensors and actuators.

This approach guarantees to generate a provably correct controller, or provide feedback if there is no controller that implements the specifications.

1.3 Contributions

This thesis proposes the solution to two major challenges which are introduced in two separated projects. In summary, the contribution includes:

- An algorithm to explore unknown regions that:
  - includes a new grammar that allows the users to specify high-level reactive tasks over potential undefined regions.
  - adjusts LTL formulas to include the new regions when the workspace is updated
  - preserve robot status and task history
- A new grammar to encode implicit memory operation by users that:
  - automatically generates extra propositions for recording event
  - automatically generates LTL formulas that change the value of the extra proposition accordingly

1.3.1 Resynthesis when Updating the Map with New Regions

This thesis presents the work for automatically generating a hybrid controller that guarantees the high-level behavior while able to relax the assumption of a known workspace [18]. An a priori completely known map is no longer an assumption; instead, the system only requires a starting map to initialize the task. The starting map is required to have enough space for containing the robot model.

As in the original approach, the desired task specification is expressed in structured English and then translated into linear temporal logic. In this work, the grammar is enriched to allow users to specify the high-level tasks over the potential unknown regions

before the execution. When the new region is detected, the system automatically assign the appropriate specifications to the new region without extra input from the users. In addition, the Robot is able to explore the workspace in either a depth-first or breadth-first strategy by choosing the priority of reordering the goal requirements, which is also indicated by the users.

When the robot is performing its task, new regions are discovered and the task is adjusted accordingly in an automatic and provable-correct manner. New regions are added into the geometric map from the actual sensor information as the first step. A re-synthesis algorithm is executed to re-write the LTL formulas for: 1) updating the accessibility information of the map in the topological space; 2) reflecting the specified tasks over the new region; 3) preserving the the completion status of the task history. Finally it re-synthesizes the LTL formulas to construct the new automaton.

The detecting process relies on a new-region sensor, the assumption of which is introduced in 3.1. The sensor used in this work is developed by Dr. Shahar Sarid. The detailed information about the sensor, regarding questions like when and how to decide a new region as well as assumptions about the working environment, are introduced in [18].

The approach is implemented within the LTLMoP toolkit[8, 16] and is demonstrated in an experiment.

1.3.2 Implicit Memory Specifications

The second challenge is to specify robot tasks that include the event memory. How to control the robot according to the memory of events? How to specify the memory related task from the user's end? These questions will be answered in this work by introducing an innovative grammar that allows implicit memory operation [17].

Since the synthesized controllers correspond to finite state machines, the robot's re-

sponses entirely depend on the current state and the environment inputs. If the events are not encoded by extra discrete variables in the state, what happened in the previous states is completely forgotten. Technically, it is possible to explicitly specify memory operations using the existing grammar introduced in [8] but it is not an ideal solution because it shifts the burden of reasoning about the memory requirements of a specification to the user, and also generates unreadable specification. The problem is described with more details in Section 3.2.

This thesis addresses the challenge of automatically managing memory operations given a specification that does not include explicit memory management. It introduces methods to automatically integrate implicit memory operations when synthesizing a finite-state robot controller from a high-level specification. The users are able to specify more intuitive structured English requirement with implicit memory where the grammar is also shorter and more readable than the original version of the structured English specification language. The memory management strategies described here are implemented within Linear Temporal Logic MissiOn Planning (LTLMoP).

1.4 Thesis Structure

The thesis is structured as follows: Chapter 2 presents background information regarding the approach and formalisms for generating high-level correct robot control, which is also a brief review of the original framework of LTLMoP. Chapter 3 defines the problems this thesis is focusing on and the assumptions that are made. Chapter 4 describes the approach to solve the problem of guaranteeing high-level reactive robot behavior while exploring a partially unknown map. An experiment is also demonstrated in the this chapter. Chapter 5 introduces the proposed new grammar for implicitly specifying memory operations, and illustrates the robot behavior produced using the described approach for a simple specification that uses memory operations. The thesis concludes in

CHAPTER 2

BACKGROUND

This section presents the background information needed for the remainder of the thesis. It includes the definition of the logical formalism, an overview of the process of generating provably-correct robot control from logical formulas and a description of LTLMoP [8], the toolbox that is used to generate the control and perform experiments.

The content in this section is also published in [18].

2.1 Linear Temporal Logic (LTL) specifications

Linear Temporal Logic (LTL) is a modal logic that includes in addition to Boolean operators (such as 'not', 'and', etc.), temporal operators, thus allowing formulas to capture truth values of atomic propositions (π) as they evolve over time.

LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following recursive rules

$$\varphi ::= \pi | \neg\varphi | \varphi \vee \varphi | \bigcirc \varphi | \Diamond\varphi | \Box\varphi \tag{2.1}$$

where $\bigcirc$ is 'Next', $\Diamond$ is 'Eventually', and $\Box$ is 'Always'.

This work considers robot specifications that are defined over a discrete abstraction of the robot motion and action and are captured in a fragment of LTL. Specifically, the atomic propositions comprise of a set $X = \{x_1, ..., x_m\}$ of environment propositions corresponding to abstract sensor information (e.g. "object detected"), and a set $Y = \{r_1, ...r_n, a_1, ..., a_k\}$ of robot propositions that correspond to the location of the robot (if r_i is true then the robot is currently in region i) and its actions a_j (e.g."flag is raised").

The fragment of LTL considered in this work follows [11, 14] where formulas are of the form $\varphi = (\varphi_e \Rightarrow \varphi_s)$; φ_e is an assumption about the sensor propositions, and thus about the behavior of the environment, and φ_s represents the desired behavior of the robot. Both φ_e and φ_s have the following structure: $\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e; \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$,

where:

- φ_i^e,φ_i^s are non-temporal Boolean formulas constraining the initial value(s) for the environment and robot respectively. They are of the form B_i, where B_i is a boolean formula over the set $X \cup Y$.
- φ_t^e,φ_t^s represents safety assumptions on the environment and safety requirements on the robot's behavior. Safety includes all behaviors that the environment or the robot must always satisfy. φ_t^e constrains the next environment state based on the current environment state and current robot state, and φ_t^s constrains the possible next robot state based on the current environment, current robot state, and the next environment state. The formulas are of the form $\Box B_i$ where B_i is a boolean formula over the set $X \cup Y \cup \bigcirc X$ for φ_t^e and $X \cup Y \cup \bigcirc X \cup \bigcirc Y$ for φ_t^s.
- φ_g^e, φ_g^s represent liveness assumptions for the environment and liveness requirements for the robot. The liveness includes goals the environment or the robot should always eventually satisfy. The formulas are of the form $\Box \diamond B_i$ where B_i is a boolean formula over the set $X \cup Y$.

One property of the synthesis algorithm is that the order of the formulas in φ_g^s determines the sequence in which the robot will fulfill its liveness requirements. For a liveness requirement φ_i, the goal number $gNum(\varphi_i) = i$ indicates that φ_i is in the i-th position in the sequence of goals. For example, in $\varphi_g^s = \Box\diamond r_a \wedge \Box\diamond(r_b \vee flag)$, the requirement that the robot eventually go to r_a is of *gNum = 0* (the first liveness) and the requirement of going to r_b or raising the flag is of *gNum = 1*.

2.2 Discrete abstraction of the workspace

The robot's workspace is assumed to be a 2 dimensional polygonal environment. The motion of the robot in the workspace is abstracted by a graph where each node represents

a region and the edges represent adjacency relations between the regions. Leveraging controllers such as those of [1, 6, 12], given a set of convex polygons, a robot can move between any adjacent regions if there are no obstacles.

For example, if the robot is able to move between two adjacent regions r_1 and r_2, the constraints on the robot motion are captured by:

$$\varphi_t^s = \begin{cases} \square(r_1 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_2)) \\ \wedge \square(r_2 \Rightarrow (\bigcirc r_2 \vee \bigcirc r_1)) \end{cases}$$

2.3 Control generation

Given a robot task as an LTL formula belonging to the fragment described above, if the task is synthesizable [16] an automaton whose behaviors satisfy the formula will be automatically generated (see [11, 14] for details).

The automaton is defined as:

$$\mathbf{A} = \{X, Y, Q, \delta, \gamma, Q_0\} \tag{2.2}$$

where

- X is the set of input (environment) propositions
- Y is the set of output (robot) propositions
- $Q \subset \mathbb{N}$ is the set of states
- $\delta : Q \times 2^X \rightarrow 2^Q$ is the labeled transition relation that maps a state and a set of environment propositions that are true to a next state
- $\gamma : Q \rightarrow 2^Y$ is the state labeling function where $\gamma(q) = y$ and $y \subseteq Y$ is the set of robot proposition that are true in state q
- $Q_0 \in Q$ the initial state

An admissible input sequence is a sequence $X_1, X_2, \ldots$ s.t. $X_j \in 2^X$ is the set of environment propositions that are true at time step j, that satisfies φ_e. A run of this automaton under an admissible input sequence is a sequence of states $q_0, q_1, \ldots$. This sequence starts at initial state $q_0 \in Q_0$ and follows the transition relation δ under the input propositions, i.e., for all $j \geq 0$, $q_{j+1} \in \delta(q_j, X_j)$.

The hybrid controller used to continuously control the robot is based on the execution of the automaton. An admissible input sequence is a sequence $X_1, X_2, \ldots$ s.t. $X_j \in 2^X$ is the set of environment propositions that are true at time step j, that satisfies φ_e. A run of the automaton under an admissible input sequence is a sequence of states $q_0, q_1, \ldots$, which starts at a possible initial state of the automaton: $q_0 \in Q_0$. At each time step, the robot sensor information is used to determine the truth values of the environment propositions X, and together with the current state q the next state q' is determined following the transition relation δ, i.e., $q' = \delta(q, X)$. γ is the state labeling function where $\gamma(q) = y$ and $y \subseteq Y$ is the set of robot proposition that are true in state q. Based on the labels of q', the next region and the next actions are performed and the appropriate low-level controllers are executed. The reader is referred to [11] for more details.

Every state $q \in Q$ has an associated goal number, which indicates the current goal (liveness requirement) the robot is heading toward. This goal number is denoted by $\gamma_r(q)$.

2.4 LTL MissiOn Planner (LTLMoP)

Linear Temporal Logic MissiOn Planning (LTLMoP) [8] is a Python-based, open-source toolkit that allows users to control physical and simulated robots by specifying high-level instructions in structured English. Note that the framework assumes perfect high-level sensors and actuators, therefore the stochastic uncertainty is not a concern of this work.

CHAPTER 3

PROBLEM STATEMENT

This chapter explains the two challenges solved by this work and introduces the assumptions of the model and environment. The two challenges are also introduced in [17, 18].

3.1 High-level Reactive behavior on Partially Known Map

Consider the case of a "Search and Rescue" scenario where a mobile robot is patrolling inside a collapsed building with unknown rooms.

Example 1 *Search and Rescue Mission*
Consider the case of a "Search and Rescue" scenario where a mobile robot is placed in a building which collapsed due to an earthquake. The robot must explore the inner parts of what is left from the building and search for survivors. The building is partially ruined, such that the original blueprint cannot be used. The robot is placed in a partially clear area, shown in Figure 3.1 (Left). There are two groups of regions: dangerous and safe. The robot is required to visit all the dangerous regions: r2,r3*, and to search for survivors (people). If the robot finds people, it will guide them back to one of the safe places:* r0 *or* r1. *If a new region is detected, it must be identified as a safe or as a dangerous region, and then be added to the map.*

The workspace is shown in 3.1. The left is the initial known map. The robot position is marked with a white circle. The right shows two new, unknown regions, *safe1, danger1*, are updated when the robot visits *r2, r3*.

The robot's high-level behavior must be guaranteed to ensure safety and mission completion while expanding the map to include unknown regions. This work focuses on guaranteeing the execution of high-level tasks by a mobile robot operating in an a priori

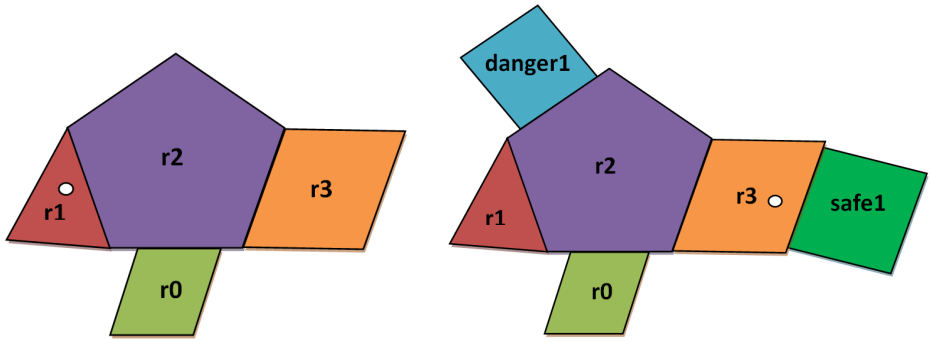


Figure 3.1: The "Search and Rescue" example.

unknown environment, and execute it's mission while detecting new regions that are not defined in its initial map.

The robot is assumed to have the appropriate sensors and actuators to perform the high-level tasks it is instructed to do. These sensors and actuators are assumed to be binary, as described in Section 2. In addition, the new region sensor sends more information regarding the new region it detected to update the geometric information of the workspace.

The basic assumptions made in this work are as follows.

Definition 1 Workspace: *The workspace is the map the robot uses, and is composed of a set of regions. The current workspace is assumed to be completely known in each step i, and is denoted* P^i*. When one new region is added into the map, the workspace is updated and the step is incremented. A workspace must have a boundary as defined in Definition 2.*

Definition 2 Boundary: *The boundary* B^i *of a workspaces* P^i *is defined as a polygon, or a set of polygons, composed of all the edges of the regions of* P^i *which are not shared between two regions. If the workspace contains holes, the boundary is composed from an outer polygon, and another polygon for each hole.*

Definition 3 New Region: *Within each step i, the workspace is assumed to be static inside its boundary. A new region* p^i *is assumed to be out of the boundary but adjacent*

to the current workspace. When a new region is detected, the previous boundary edge shared between the new region and the known region becomes a transition edge.

Definition 4 New-region Sensor: *The robot must have the capabilities to detect new regions. The sensor is required to return two types of information: 1) a binary variable indicating if a new region is detected; 2) the detailed geometric information of the new region. There is no other limitation to the sensor or the detecting algorithm. For the experiments reported in this paper, a laser scanner was used.*

To ensure safety and correctness of the high-level robot behavior, it is impossible to plan ahead without knowing what is ahead. The preferred approach for a high-level task is to map the environment prior to generating the controller. Whenever the map is changed, a new controller is generated accordingly. The expansion of workspace is defined as followed.

Definition 5 Expansion: *The workspace can be dynamically expanded in each step. The robot starts in step* 0 *with initial workspace* P^0. *In each step i, a new region* p^i *is added. Each* p^i *must be adjacent to at least one known region. The workspace is recursively defined as follows,* $P^i = P^{i-1} \cup p^i$, $\forall i > 0$.

There are several challenges to be solved :

Problem 1 *Operating in an unknown environment:*

- *Given an initial known workspace,* P^0, *how to specify high-level tasks over the potential unknown region?*
- *Given a mobile robot equipped with a new-region sensor, at each time step i when a new region* p^i *is added to the current workspace* P^i, *how to adjust the controller to include the new region?*

- *Each time the controller is adjusted, how to preserve the history of task completion?*

3.2 Memory in Execution

Consider the requirement, "Every time an order is made, go to the kitchen". In the structured English language defined in [11], the closest match to this specification is "If order then go to kitchen", which in turn corresponds to the LTL formula $\square\diamondsuit(\pi_{order} \implies \pi_{kitchen})$. The symbol $\square\diamondsuit$ implies a liveness requirement in the sense that the formula $\pi_{order} \implies \pi_{kitchen}$ has to be eventually true in some future time steps.

The LTL formula aims to ensure that if the robot senses π_{order} in a single time-step, it will eventually visit the kitchen to fulfil the task. However, the semantics may easily fail in such circumstance: if π_{order} is true in a single time-step, but not in any subsequent ones, the above liveness requirement is still satisfied because the semantics of this LTL formula require that either $\pi_{kitchen}$ holds infinitely often or $\neg\pi_{order}$ holds infinitely often. Therefore whether or not the robot goes to the kitchen, it doesn't violates the LTL specification.

As a result, the synthesized automaton may not include any states in which the robot is in the kitchen. In a physical experiment, if the customer signals the end of an order, the robot will sense π_{order} and begin moving to the kitchen. However, if the robot loses sight of the customer on the way to the kitchen, it will no longer sense π_{order}, and will therefore forget the original purpose.

This observation motivates the use of implicit memory for each relevant event. As with everything else in the discrete problem abstraction, this memory will be represented using logic propositions that are set on the associated event.

Note that it is possible to modify the above specification using the original grammar in [11] to explicitly specify changes in memory by introducing a new proposition m_{order}

(and the corresponding Structured English phrase "memo_order"). Listing 1 shows the additional structured English and corresponding LTL that accomplishes this.

Listing 1 Specification demonstrating the additional specification sentences required to avoid forgetfulness

Do memo_ order if and only if you are sensing order
or you were activating memo_ order
$\square(\bigcirc m_{order} \iff (\bigcirc \pi_{order} \vee m_{order}))$
If you are activating memo_ order then visit kitchen
$\square\diamondsuit(m_{order} \implies \pi_{kitchen})$

As shown above, the grammar introduced in [11] requires the user to explicitly introduce an extra propositions m_{order} and specify the events that cause it to change the value. More specifically, the user would have to include the sentence "Do memo_ order if and only if you are sensing order" in the specification above. This places the burden of reasoning about what memory needs to be recorded on the user.

Another example is the repeating of goal requirement. Consider the LTL formula $\square\diamond(check\_in\_desk)$, which will drive the robot to visit the destination repeatedly during the execution. But if the robot doesn't need to repeat check-in, the correct LTL formula would be:

$$\square(\bigcirc m\_desk \iff (m\_desk \vee \bigcirc \pi_{check\_in\_desk})$$

$$\wedge\square\diamond(m\_desk)$$

The first line of the formula expresses that if the event $\pi_{check\_in\_desk}$ turns true, the memory m_desk turns true and stays true forever. And the second line expresses that the robot is required to eventually make the memory turn true by visiting the $\pi_{check\_in\_desk}$ at least once. Again the memory operation is necessary to ensure the correct behaviors.

The work in this thesis aims to solve the problem that:

Problem 2 *Implicit Memory Operation*

- *Enrich the structured English specification grammar with constructs that allow unambiguous but implicit memory operations.*
- *Given a specification S in this enriched grammar, automatically generate a set of memory propositions M and the set of LTL formulas Φ corresponding to the implicit memory operations specified.*

CHAPTER 4

GUARANTEED HIGH-LEVEL BEHAVIOR WHILE EXPLORING PARTIALLY KNOWN MAP

This section introduces the new grammar for specifying high-level task over the potential new regions and describes the algorithms to automatically re-synthesize the controller when a new region is added to the partially known workspace. The algorithm preserves the history of task completion, assigns the high-level task over the new region and re-generates the controller. An experiment is also introduced to illustrate the proposed method.

The content of this section has been published in [18].

4.1 Grammars

The first question to answer is: How to specify the robot task over the undefined regions?

This thesis proposes using region groups and quantifiers to allow users to specify tasks over multiple regions, including the unknowns. The grammar for activating the re-synthesis algorithm and assigning the new region to the existing region group is also introduced. The original grammar [8, 11] used by LTLMoP has been enriched with quantifiers and reactions to new regions needed for defining tasks in unknown maps.

4.1.1 Quantifiers

The content in this subsection is Cameron Finucane's contribution. Thanks to Cameron for developing this extremely important capability that built the foundation of this work.

Consider the scenario in Example 1, the same task is assigned over multiple regions. When a new region is detected, no matter safe or dangerous, extra specification is needed for defining the tasks over the new region. Therefore an automatic process for assigning

tasks over multiple regions is necessary.

To deal with such conditions, quantifiers are introduced in this section. To apply quantifiers over multiple regions, one defines groups as follows:

- Group *groupName* is *region1, region2, region3...*

If the *groupName* is used together with the quantifiers "all" or "any", the sentences are automatically converted into LTL formulas over the regions. The translation process is as in Algorithm 1.

Algorithm 1 Translate the structured English with quantifier

```
1: regionList ⇐ regions in groupName
2: translate spec. to LTL (groupName stays as temp. proposition)
3: for i ∈ [1, len(regionList)] do
4:     if quantifier 'any' then
5:         result_regions ⇐ join regionList[:] with ∨
6:         φ ⇐ substitute groupName with result_regions
7:     if quantifier 'all' then
8:         results[i] = substitute groupName with regionList[i]
9:         φ ⇐ join results[:] with ∧
```

Revisiting Example 1, the new specification would be:

- Group *Safe* is r_0, r_1
- Group *Dangerous* is r_2, r_3
- If you are not activating *guide* then visit all *Dangerous*
- If you are activating *guide* then visit any *Safe*

which translates into

$$\wedge_{i\in\{2,3\}} \square\diamond((\neg a^{guide}) \Rightarrow r_i)$$

$$\wedge \square\diamond((a^{guide}) \Rightarrow (r_0 \vee r_1))$$

These quantifiers facilitate writing specifications for unknown workspaces as they do not require the explicit enumeration of all regions. When a new region is detected, the

user is not required to manually write the additional specification and the robot is able to automatically re-synthesize the controller and resume the execution. The grammar for re-synthesis is introduced in Section 4.1.2, and the algorithm is introduced in Section 4.2.

4.1.2 Specification for re-synthesis

Now assume, in Example 1, when the robot enters *r2*, it detects a new dangerous region *danger1* adjacent to *r2*. Then, when it enters *r3*, it detects a new safe region *safe1* adjacent to *r3*. The new workspace is shown in Figure 3.1 (Right).

A special robot action "re-synthesize" is defined, which terminates the execution of the hybrid controller and re-generates the controller. "re-synthesize" is used in the same manner as a robot proposition in the requirements specification:

- If you are sensing *regionSensor* then do re-synthesize.

The re-synthesize action is activated when the *regionSensor* returns true.

If the user explicitly indicates which group to add the new region to, the extra indication is allowed as follows:

- If *regionSensor* then do re-synthesize and add into *groupName*.

The following relation is defined: *regionSensor* $\rightarrow$ *groupName*. Later, during the re-synthesis process, if *regionSensor* = *True*, the corresponding *groupName* is returned to ensure that the correct group is updated with the new region. Furthermore, only specifications applied to this group will need to be rewritten as new LTL formulas. If the robot is capable of distinguishing between different features of the new regions, the detected regions can be added into different groups accordingly. In this work, each region group is represented by a single new-region sensor, as in Example 1:

- If you are sensing safe-new-region then do re-synthesize and add it into Safe.
- If you are sensing dangerous-new-region then do re-synthesize and add it into Dangerous.

4.2 Controller re-synthesis during execution

This section describes the algorithms and automated process for automatically re-synthesizing the high-level controller when new areas are found. It addresses the special robot action *re-synthesize*, the need for capturing the current robot state and goal, the detection of a new region using sensors, the process of modifying the discrete abstraction of the workspace and the algorithm for creating and synthesizing a modified LTL formula.

Recall the automaton defined in Equ. 2.2. The re-synthesis process generates a new automaton:

$$\hat{\mathbf{A}} = \{X, \hat{Y}, \hat{Q}, \hat{\delta}, \hat{\gamma}, \hat{q}_0\}$$

Note that the environment proposition set X is not changed, and the new robot proposition set $\hat{Y}$ is defined by adding the region proposition *r_new-region* to the original Y. The other functions of the new automaton $\hat{Q}, \hat{\delta}, \hat{\gamma}, \hat{\gamma}_r$ are also achieved by modifying the LTL formulas φ_e, φ_s. This section presents the algorithm which determines the new automaton $\hat{\mathbf{A}}$.

4.2.1 The robot action "re-synthesize"

Detection of new region is captured by a Boolean environment proposition, *regionSensor*. The re-synthesis action is captured by a robot proposition. The two propositions are denoted as $s^{new-region}$ and $a^{re-synthesize}$ respectively. The specification "If you are

sensing *regionSensor* then do re-synthesize" is captured by the LTL formula:

$$\square(\bigcirc s^{new-region} \Rightarrow \bigcirc a^{re-synthesize}) \tag{4.1}$$

The "re-synthesize" action is a special action since the low-level controller associated with it terminates the execution of the automaton, and calls the module to rewrite the LTL formula and re-synthesize an appropriate automaton.

The re-synthesis process is shown in Algorithm 2. In line 1, the execution of the current automaton is terminated. The following two propositions are reset: $s^{new-region} = False$ and $a^{re-synthesize} = False$. In lines 2-3 the environment propositions and robot propositions of the current state are recorded, in order to describe the initial state of the robot when resuming the execution. In line 4, the initial condition for the new automaton is obtained and the LTL formulas φ_i^s, φ_i^e for initial conditions are updated. In line 5, the new region proposition is added as: $\hat{Y} = Y \bigcup r_{new}$. The process of modifying the workspace and rewriting the LTL formula φ_t^s is discussed in Section 4.2.2. In line 6, the goal number of the liveness requirement toward which the robot is currently moving is recorded. In line 7, re-ordering of the goal requirements is achieved by rewriting the LTL formula φ_g^s as described in Algorithm 3 and Section 4.2.3. In lines 8-9, the updated LTL formula is synthesized and the new automaton is executed.

Algorithm 2 Low-level controller for the *re-synthesize* action

1: Break execution, reset $s^{new-region}$, $a^{re-synthesize}$
2: CurrRobotState $\Leftarrow \gamma(q)$
3: CurrEnvState $\Leftarrow$ values of sensor propositions
4: NewInitState: $\hat{q}_0 \Leftarrow$ CurrRobotState $\wedge$ CurrEnvState
5: Modify discrete abstraction in workspace, get $\hat{Y}$ (see Section 4.2.2)
6: CurrGoalNum $\Leftarrow \gamma_r(q)$
7: Modify liveness conditions in LTL (see Algorithm 3)
8: Re-synthesize the automaton
9: Load new automaton and resume execution

4.2.2 Modifying the discrete abstraction

As defined in Section 3.1, the robot maintains a map of the workspace and expands it on-the-fly. The map maintained by the robot is composed of a list of polygonal regions. At step i, the robot is equipped with a map of the already known area, P^i, as well as its outer boundary, B^i.

When a new region is detected by the sensor, the sensor returns the geometric information of the new region. The topological information of the new region is then extracted, the transition relationship between the new region and the known regions is determined. The constraint of robot motion is obtained by re-writing the robot safety assumption φ_t^s.

Revisiting Example 1, the change in the known workspace results in an updated φ_t^s:

$$\begin{cases} \square(r_0 \Rightarrow (\bigcirc r_0 \vee \bigcirc r_2)) \\ \wedge \square(r_1 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_2)) \\ \wedge \square(r_2 \Rightarrow (\bigcirc r_2 \vee \bigcirc r_0 \vee \bigcirc r_1 \vee \bigcirc r_3 \vee \bigcirc danger_1)) \\ \wedge \square(r_3 \Rightarrow (\bigcirc r_3 \vee \bigcirc r_2 \vee \bigcirc safe_1)) \\ \wedge \square(danger_1 \Rightarrow (\bigcirc danger_1 \vee \bigcirc r_2)) \\ \wedge \square(safe_1 \Rightarrow (\bigcirc safe_1 \vee \bigcirc r_3)) \end{cases}$$

4.2.3 Rearranging the liveness conditions

Given the current goal number, the robot is able to determine which liveness requirement it was pursuing. Therefore, it is able to distinguish between the complete and incomplete goals. The history of the completed high-level tasks is captured by re-assigning the order of the goals. The robot is allowed to first address incomplete liveness requirements, and in addition, to choose the exploration strategies by inserting the new goal at different positions, either first, thereby creating a depth first strategy, or afterwards, creating a breadth first strategy.

Algorithm 3 Rewriting the LTL formula for liveness requirements

1: CompGoals $\Leftarrow \varphi \in \varphi_g^s$, $gNum(\varphi) <$ CurrGoalNum
2: IncompGoals $\Leftarrow \varphi \in \varphi_g^s$, $gNum(\varphi) \geq$ CurrGoalNum
3: *groupName* $\leftarrow$ *regionSensor*
4: Translate specs with *groupName* into LTL
5: newLiveness $\Leftarrow$ liveness with newRegion in new LTL
6: **if** Depth First Order **then**
7: $\quad \varphi_g^s \Leftarrow$ newLiveness $\wedge$ IncompGoals $\wedge$ CompGoals
8: **if** Breadth First Order **then**
9: $\quad \varphi_g^s \Leftarrow$ IncompGoals $\wedge$ newLiveness $\wedge$ CompGoals

The detailed process is shown in Algorithm 3. In lines 1-2, the liveness requirements are classified as complete or incomplete goals. In line 3, the user's predefined group for the new region is loaded. In lines 4-5, the liveness with the relevant *groupName* is translated into LTL, as described in Section 4.1.1. In lines 6-8, the goals are re-ordered, if following the Depth First strategy, the new goal is put before incomplete goals. In lines 9-11, if following the Breadth First strategy, the new goal is added after the set of incomplete goals.

Revisiting Example 1, assume that when the new region *danger1* is detected, the robot has already visited r_1, r_2, so the priority of the original goals has changed. The robot is also capable of choosing between either a depth-first order or breadth-first order, which is achieved as follows:

Depth First(Alg 3, ln 6-8):	Breadth First(Alg 3, ln 9-11):
$\varphi_g^s =$	$\varphi_g^s =$
$\Box \diamond ((\neg a^{guide}) \Rightarrow danger_1)$	$\Box \diamond ((\neg a^{guide}) \Rightarrow r_3)$
$\wedge \Box \diamond ((\neg a^{guide}) \Rightarrow r_3)$	$\wedge \Box \diamond ((\neg a^{guide}) \Rightarrow danger_1)$
$\wedge \Box \diamond ((\neg a^{guide}) \Rightarrow r_2)$	$\wedge \Box \diamond ((\neg a^{guide}) \Rightarrow r_2)$
$\wedge \Box \diamond ((a^{guide}) \Rightarrow (r_0 \vee r_1))$	$\wedge \Box \diamond ((a^{guide}) \Rightarrow (r_0 \vee r_1))$

4.3 Experiments

The work described in this thesis was implemented on a physical robot in the lab, where the on-board laser range finder was used to detect previously unknown regions while the robot was performing a high-level tasks. The scanner was used to detect the free space outside the known map's boundary, and thus served as a new-region sensor.

Thanks to Dr. Shahar Sarid for setting up this experiment in both hardware and software aspects. Please refer to [18] for detailed information about the experimental set-up.

4.3.1 Description of the Scenario

Example 2 *Classroom Assistant*

The robot is looking for students who need to submit assignments to their professors in the workspace depicted in Fig. 4.1. The robot is wandering through the classrooms until it finds such a student. Once it is handed an assignment, it searches for a professor in the offices until it finds one. If a door opens (new classroom or new office), the robot explores that region, classifies it, and continues executing its task accordingly. If the robot detects a hazardous item in front of classroom2, it avoids entering that classroom and it raises its flag as warning. The specifications, given in structured English, are presented in Listing 2.

The map of the workspace is shown in Figure 4.1. The left figure shows that the initial map is composed of *hall1, hall2, classroom1, classroom2, office1*. The right figure shows the top view of the experiment field. The robot starts with a map including 5 known regions. Two groups are defined: Classrooms and Offices. If there is no assignment, the robot must continuously visit all Classrooms, and if there is it should visit all Offices. If the robot senses a hazardous item it must avoid classroom2 and raise the flag. If a new region is sensed, the robot must stop, do re-synthesize and add it to

Listing 2 Specification for the experiments.

```
1  Environment starts with false
2  Robot starts with false
3  Always not (newClassroom and newOffice)
4  Always not ((newClassroom or newOffice) and hazardous)
5  Group Classrooms is classroom1
6  Group Offices is office1
7  Do flag if and only if you are sensing hazardous
8  If you are sensing newClassroom then do re-synthesize and add to Classrooms
9  If you are sensing newOffice then do re-synthesize and add to Offices
10 If you are sensing newClassroom or you are sensing newOffice then stay
11 If you are not sensing newClassroom and you are not sensing newOffice then do not
       re-synthesize
12 If you are not sensing assignment and you are not activating re-synthesize then
       visit all Classrooms
13 If you are sensing assignment and you are not activating re-synthesize then visit
       all Offices
14 If you are not sensing hazardous and you are not sensing assignment and you are not
        activating re-synthesize then visit classroom2
15 If you are sensing hazardous then always not classroom2
```

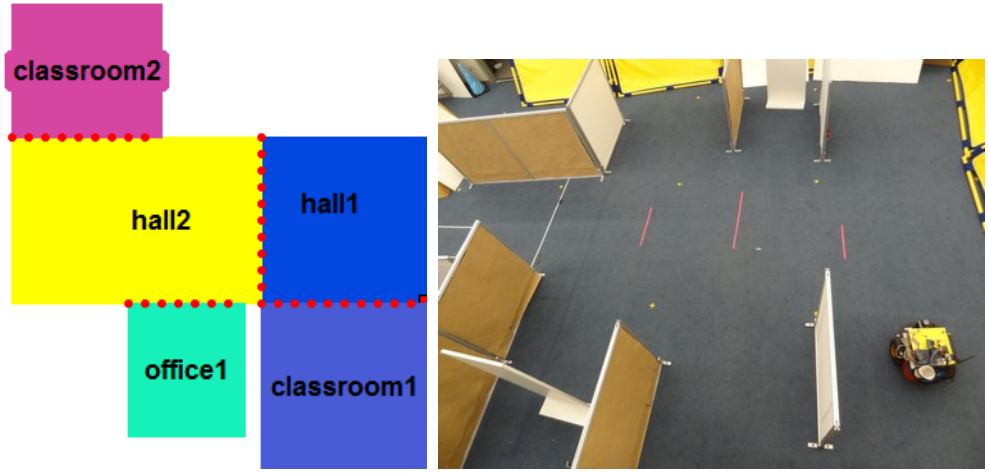


Figure 4.1: The initial map before execution.

the relevant region group.

4.3.2 Local Sensors and Actuators

The robot sensors and actuators are explained as followed:

Sensors: newClassroom, newOffice, hazardous, assignment

- **newClassroom/newOffice**: The two propositions are attached to the same new-region sensor. If the detected new region is larger than the threshold, the sensor returns true in **newClassroom**. Otherwise **newOffice** returns true.
- **hazardous**: A red cone served as a hazardous item. When the cone is detected by the robot, the proposition turns true.
- **assignment**: The object sensor, which is a cup that can sense the presence or absence of objects within, was used to sense the presence of an assignment.

Actuators: flag, re-synthesize:

- **flag**: The robot action is raising a flag if the proposition value is true, or lowering a flag if false.
- **re-synthesize**: When the proposition is true, it activates the local controller that terminates the execution and starts the re-synthesis process.

Figure 4.2 shows some of the items described above. The cone represents a hazardous item in front of a classroom and the raised flag was the robot's response to detecting the cone. The cup contained a touch sensor and was used to collect assignments.

4.3.3 Description of the Experiment

The robot starts in *classroom1*, and heads toward *hall1*. As soon as it enters *hall1*, it detects *new1* (Fig. 4.3a,4.3b) and classifies it as a large room, thus adding it to the group *classrooms* and then re-synthesizing. It visits region *new1* and *classroom1* according to the specifications, then it heads toward *classroom2*. On its way, the robot passes through

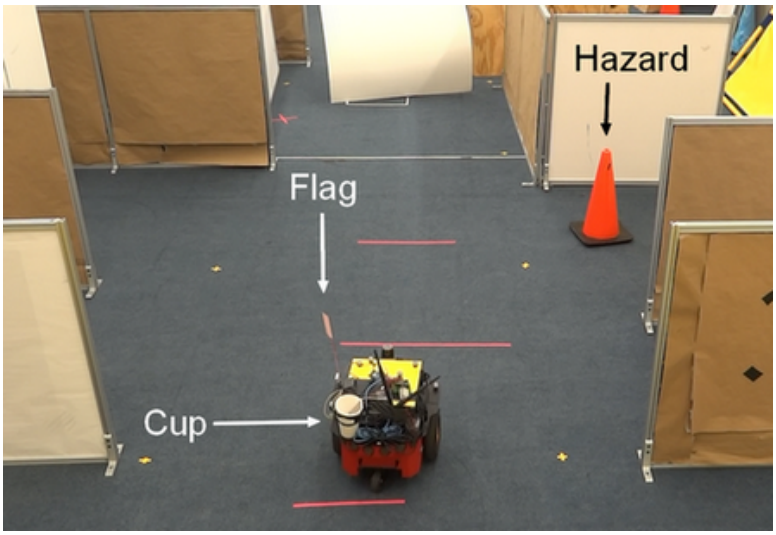
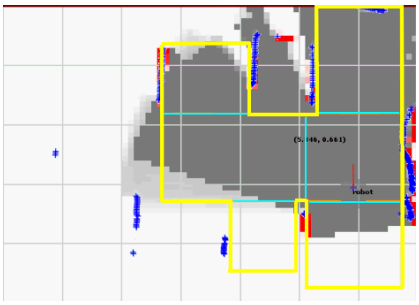


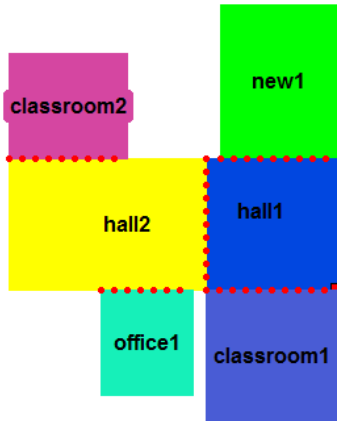
Figure 4.2: A snapshot of some hardwares

hall2, where it detects *new2* (large) (Fig. 4.3c,4.3d), adds it to group *classrooms*, and re-synthesizes. Immediately afterwards, it detects *new3* (small) (Fig. 4.3c,4.3d), adds it to group *offices* and re-synthesizes. Afterward, it continue to classroom *new2*, there, a student hands the robot an assignment (Fig. 4.3e), and the robots start searching for the professor to submit the assignment to in the offices *new3* and *office1*. Since the robot does not find the professor in those offices, it continues to search the offices until a door to a new office is opened, the robot detects *new4* (Fig. 4.3c,4.3d), adds it to group *offices* and re-synthesizes. The robot moves to the *new4* office, there it finds the professor and hands him the assignment (Fig. 4.3f). Since now the robot does not have assignments to deliver, it continues to search for students in the classrooms, starting with *classroom2* which it did not visit, yet.

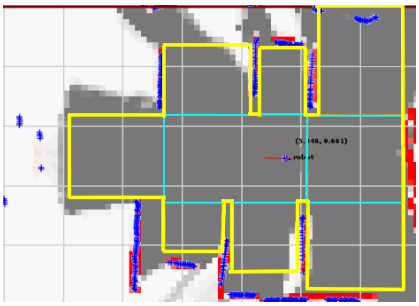
The robot continues its correct execution according to the specifications, continuously searching for students with assignments and handing them to professors. After some time, a hazardous item appears in front of *classroom2* (Fig. 4.3g). When the robot is in *hall2* and heading to *classroom2*, it detects the hazardous item, raises the warning flag (Fig. 4.3g), skips *classroom2* and continues to *new2* instead. After the hazardous



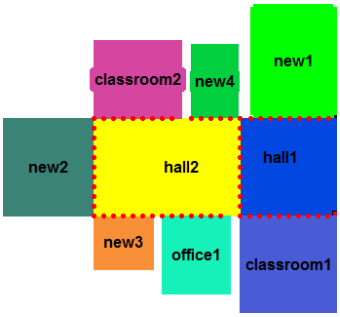
(a) After detecting the classroom *new1*. The boundary is in yellow, the regions are separated by turquoise lines.



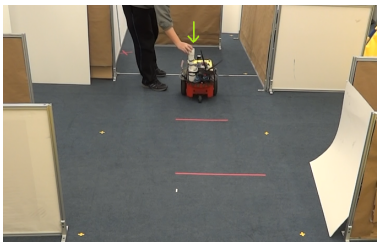
(b) LTLMoP map after adding classroom *new1*.



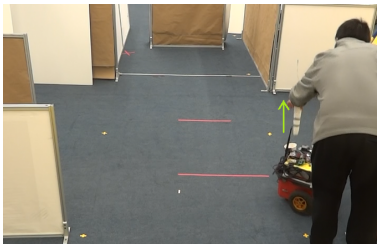
(c) After adding classroom *new2* and offices, *new3,new4*.



(d) LTLMoP map after the addition of the office *new4*.



(e) Student submit assignment to robot in *classroom2*.



(f) Robot submit assignment to professor in office *new4*.

(g) Avoiding *classroom2* and raising the warning flag as a response to the hazardous cone.

(h) Visiting the office *new3* with assignment in the cup

Figure 4.3: Assignment collecting and handing experiment

item exits the view of the sensor, the robot lowers the flag, and continues execution with correct behavior.

From the experiment execution, it is evident that under the assumptions of Section 3.1, correct behavior is guaranteed even when adding regions that are unknown a priori. Moreover, the robot correctly classified the new regions to *offices* and to *classrooms* according to their sizes, and added them to the specifications accordingly. A video is provided on: http://www.youtube.com/watch?v=oiF4Wt8xgio&feature=plcp

CHAPTER 5

MEMORY

This section introduces the enriched grammar that allows implicit memory operations by users, which extends the work in [11]. The grammar automatically generates extra propositions for recording events as well as LTL formulas for changing their values accordingly. Users are allowed to use the grammar for both robot and environment assumptions.

The content in this section has been published in [17].

5.1 Memory Proposition

To record the memory, a new class of propositions is introduced and defined as followed.

Definition 6 Memory Proposition: *A memory proposition is a Boolean proposition m_ϕ whose value corresponds to the occurrence of event ϕ.*

The purpose of memory propositions is to record that a specific event has occurred in the execution. Once it becomes true, the memory proposition stays true until its resetting condition (if any) is met. The basic structure of a memory proposition without a resetting condition is:

$$\square(\bigcirc m\_\phi \Leftrightarrow (\bigcirc \phi \vee m\_\phi))$$

Once ϕ is true, the memory proposition m_ϕ turns true and stays true to record this event. In the example in Section 3.2, m_{order} is a memory proposition whose value responds to π_{order}.

5.2 Structured English Grammar

The syntax and semantics of the new grammar constructs are listed in Table 5.1. The constructs are grouped into Types, labeled by the first column of the table.

5.2.1 Regular Types

Types 1-4 define the four basic types of implicit memory propositions, classified based on the events remembered. Types *1-*5 introduce derived memory operations that use the four basic types; details are described in Section 5.2.2.

The second column of the table explains the events to be remembered. Types 1-2 are complementary – Type-1 propositions remember that a condition has been satisfied, while Type-2 propositions remember that a requirement has been fulfilled. Type-2 propositions are useful for specifying non-repeated goals. For example, $\Box \diamond (\pi_{region})$ will drive the robot to visit the *region* infinitely often, while $\Box((\bigcirc m\_r) \Leftrightarrow (m\_r \vee \bigcirc \pi_{region})) \wedge \Box \diamond (m\_r)$ will not result in repeating behavior. Type-3 propositions remember that a requirement has been fulfilled under a specific condition; this allows specifying the same requirement for multiple conditions. Type-4 propositions are like Type-1 propositions, but can be set back to false by a second condition ϕ_2. Note that if both conditions are true, the corresponding Type-4 proposition will be set to false.

The third column in the table lists the admissible structured English grammar corresponding to each type of proposition. The symbol Θ represents structured English phrases of the form,"you are activating *action*", "visit *region*", etc, that conform to the grammar described in [11]. The composition of multiple phrases connected by "and/or" is also acceptable. Furthermore, this work extends the admissible form of Θ to include perfect tense phrases such as "you have sensed/activated/visited *sensor*/*action*", which correspond to the events remembered by Type-1 and

Type-4 propositions. Note that in Type-2 and 3, the short phrase "at least once" is optional if following "visit/go to", and therefore "visit π_{region}" is now translated into $\square((\bigcirc m\_r) \Leftrightarrow (m\_r \vee \bigcirc \pi_{region})) \wedge \square \diamond (m\_r)$ in contrast with the definition in [11]. The new grammar for repeatedly visiting a region is "Repeatedly visit/go to *region*".

The fourth column lists the equivalent LTL formula for each sentence in the previous column. The symbol ϕ is a Boolean formula with recursive form $\phi ::= \pi | \neg\phi | \vee\phi | \wedge\phi$. The structured English phrase Θ_{name} corresponds to the LTL formula ϕ_{name}. The symbol Δ can represent both $\square$ and $\square\diamondsuit$.

Note that Type-1 propositions impose different grammatical constraints on safety and liveness requirements to reduce the ambiguity of the structured English. The other types distinguish between safety and liveness requirements based on keywords in Θ. Following the grammar in [11], "do/activate/sense/in π" implies a safety requirement, and "repeatedly visit/infinitely do π" implies a liveness requirement.

Type	What to remember?	Structured English ($\mathscr{S}$)	LTL ($\mathscr{M}, \Phi$)
1	Condition has happened	Once Θ_{cond} then Θ_{req_safe} from now on	$\Box(m\_\phi_{cond} \Rightarrow \phi_{req\_safe}) \wedge$
			$\Box(\bigcirc m\_\phi_{cond} \Leftrightarrow (\bigcirc \phi_{cond} \vee m\_\phi_{cond})$
		After Θ_{cond} then Θ_{req_live} repeatedly	$\Box \Diamond (m\_\phi_{cond} \Rightarrow \phi_{req\_live}) \wedge$
			$\Box(\bigcirc m\_\phi_{cond} \Leftrightarrow (\bigcirc \phi_{cond} \vee m\_\phi_{cond})$
2	Requirement has happened	Θ_{req} (at least once)	$\Box \Diamond (m\_\phi_{req})$
			$\Box(\bigcirc m\_\phi_{req} \Leftrightarrow (\bigcirc \phi_{req} \vee m\_\phi_{req})$
3	Requirement has happened	While Θ_{cond} then Θ_{req} (at least once)	$\Delta(\phi_{cond} \Rightarrow m\_\phi_{cond} \phi_{req})$
	under certain condition		$\Box(\bigcirc m\_\phi_{cond}\_\phi_{req} \Leftrightarrow ((\bigcirc \phi_{req} \wedge \phi_{cond}) \vee m\_\phi_{req}))$
4	Memo is set on Θ_1 and reset on Θ_2	After/once Θ_1 then Θ_{req} until Θ_2	$\Delta(m\_\phi_1 \phi_2 \Rightarrow \phi_{req})$
			$\Box(\bigcirc m\_\phi_1 \phi_2 \Leftrightarrow ((\bigcirc \phi_1 \vee m\_\phi_1 \phi_2) \wedge \neg \bigcirc \phi_2)$
*1	'Only'+cond	Only once Θ_{cond} then Θ_{req_safe} from now on	LTL in Type 1 +
		Only after Θ_{cond} then Θ_{req_live} repeatedly	$\Box((\neg \bigcirc m\_\phi_{cond}) \Rightarrow (\neg \bigcirc \phi_{req}))$
*2	requirement + 'only once'	Eventually Θ_{req_live} only once	LTL in Type 2 + $\Box(m\_\phi_{req} \Rightarrow (\neg \bigcirc \phi_{req}))$
*3	requirement under condition + 'only once'	If Θ_{cond} then eventually Θ_{req_live} only once	LTL in Type 3 + $\Box(m\_\phi_{cond}\_\phi_{req} \Rightarrow (\neg \bigcirc \phi_{req}))$
		If Θ_{cond} then Θ_{req_safe} only once	
*4	Memo is self-reset when the requirement is met	After each time Θ_{cond},	$\Delta(m\_\phi_{cond} \phi_{req} \Rightarrow \phi_{req})$
		Θ_{req} (at least once)	$\Box(\bigcirc m\_\phi_{cond} \phi_{req} \Leftrightarrow ((\bigcirc \phi_{cond} \vee m\_\phi_{cond} \phi_{req}) \wedge \neg \bigcirc \phi_{req})$
*5	Condition-Requirement memos on both sides	After the first time Θ_{cond},	$\Box(\bigcirc m\_\phi_{cond} \Leftrightarrow (\bigcirc \phi_{cond} \vee m\_\phi_{cond}))$
		Θ_{req} (at least once)	$\Box(\bigcirc m\_\phi_{cond}\_\phi_{req} \Leftrightarrow ((\bigcirc \phi_{req} \wedge \bigcirc m\_\phi_{cond}) \vee m\_\phi_{req})$
			$\Delta(m\_\phi_{cond} \Rightarrow m\_\phi_{cond}\_\phi_{req})$

Table 5.1: Syntax and Grammar

5.2.2 Derived Types

In Types *1-*3, the modifier "only" is introduced to constrain either the condition or the requirement. If the keyword "only' modifies the condition ('only'+ $< cond >$), it requires the robot to perform the request only after the condition has happened. On the

other hand, if the keyword modifies the requirement ($< req >$ + 'only once'), it requires that the robot fulfil the requirement only once, and prohibits it from being performed again.

In Type 1, once the condition has happened, the memory proposition becomes true and stays true forever. Type *4 introduces a variation with self-resetting of the memory proposition: this is a special case of Type-4 propositions, and allows specifying scenarios in which the user requires the robot to fulfil a requirement each time the condition is true.

Finally, Type *5 propositions combine Types 1 and 3, and are introduced when the specification directs the robot to perform a requirement at least once if the condition has happened.

5.3 Example

The following example illustrates a representative high-level robot task and the associated discrete abstraction.

Example 3 *Consider a robot waiting tables at a restaurant. Its job description for a single day is as follows. At the beginning of the day, the robot enters the restaurant and goes straight to the check-in-desk. It greets the first shipping truck of the day at the loading dock (but need not worry about subsequent incoming trucks). It is required to put on a waiter's tuxedo only after having met the truck. When customers arrive, the robot moves between the three dining rooms to wait for an order. Every time an order is made, it goes to the kitchen to places the order with the chefs. The restaurant map is depicted in Fig. 5.1.*

The robot has three sensors, which sense shipping trucks, customers and the completion of an order – it is assumed that the customer will signal the end of their order to the

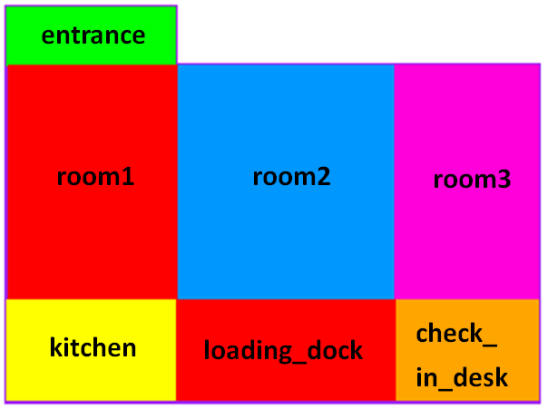


Figure 5.1: Workspace for Example 3

robot. These sensors correspond to propositions π_{truck}, $\pi_{customer}$ and π_{order} respectively. The robot also has one action in addition to motion, which is putting on its tuxedo, π_{wear_tux}. In addition, there are seven possible locations: the entrance, check_in_desk, loading_dock, kitchen, and three dining rooms (corresponding to $\pi_{entrance}$, π_{desk}, π_{dock}, $\pi_{kitchen}$, π_{room_1}, π_{room_2}, and π_{room_3}, respectively.

The robot task can be precisely defined using LTL formulas over this set of propositions. For example, the requirement "always wear a tuxedo" would translate to $\Box\, \pi_{wear\_tux}$. Additional formulas constrain the possible motion of the robot in the workspace, as determined by the topological constraints, and to account for the fact that the robot can be in exactly one location at any given time.

The task specification described in Example 3 in natural language contains sentences requiring the robot to remember events and, if necessary, to later forget them in order to react to subsequent events. Using the proposed (enriched) grammar, the specification is captured in Listing 3.

This concise, intuitive specification illustrates the power of the proposed enriched structured English grammer. The memory propositions created by the specification are: m_{check_in}, m_{truck}, m_{dock}, $m_{customer,order}$, $m_{order,kitchen}$.

Listing 3 English specification with implicit memory

```
1  Go to check_in_desk.
2  After the first time you have sensed truck, go to loading_dock.
3  Only once you have visited loading_dock then do wear_tux from now on.
4  Group dining_rooms is room1, room2, room3
5  After you have sensed customer then visit all dining_rooms until you are sensing
       order.
6  After each time you have sensed order, go to kitchen.
```

In comparison, the specification without implicit memory propositions (using the grammar in [11]) would be longer and far less readable. Listing 4 shows the specification equivalent to Listing 3, using the original grammar in [11].

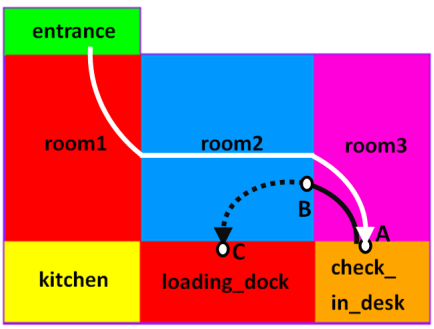
Listing 4 Original English specification equivalent to Listing 3

```
1  Do memo_check_in if and only if you are in check_in_desk or you were activating
       memo_check_in
2  Repeatedly visit memo_check_in
3  Do memo_truck if and only if you are sensing truck or you were activating
       memo_truck
4  Do memo_dock if and only if you are in loading_dock or you were activating
       memo_dock
5  If you are activating memo_truck then visit memo_dock
6  If you are activating memo_dock then do wear_tux
7  If you are not activating memo_dock then do not wear_tux
8  Do memo_customer if and only if (you are sensing customer or you were activating
       memo_customer) and you are not sensing order
9  Group dining_rooms is room1, room2, room3
10 If you are activating memo_dock then visit all dining_rooms
11 Do memo_order if and only if (you are sensing order or you were activating
       memo_order) and you are not in kitchen
12 If you are activating memo_order then visit kitchen
```

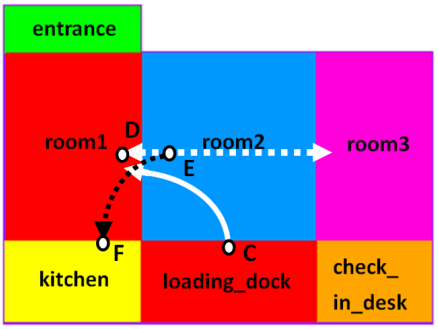
5.4 Demonstration of the Example

This section depicts a simulation of the robot behavior synthesized in LTLMoP using the specification in Listing 3. The motion specified by Lines 1-2 of the specification is shown in Fig. 5.2a, and that of Lines 5-6 is shown in Fig. 5.2b.

As in the figures, the robot motion satisfies the specification in Listing 3. The curves represent the robot motion; the points represent occurrence of the events which cause



(a) Motion of Lines 1 (white) and 2 (black)



(b) Motion of Lines 5 (white) and 6 (black)

Figure 5.2: The demonstration of Example 3

.

changes in the values of memory propositions; the dotted curves represent the robot motion after the events ended while the memory propositions still stay true.

In Fig. 5.2a, Line 1 is shown by the white curve: the robot starts at the **entrance** and moves to **check_in_desk**. At point **A**, the memory proposition m_{check_in} turns true and stays true, fulfilling thus the corresponding liveness requirement. Line 2 is shown by the black curve: the robot moves to the **loading_dock** after it senses a **truck**. At point **A**, the sensor proposition π_{truck} and the memory proposition m_{truck} become true, and the robot moves towards **loading_dock**. At point **B**, π_{truck} turns false while m_{truck} stays true; this doesn't influence the robot motion, as shown by the dotted curve. At point **C**, the memory proposition m_{dock} turns true and stays true.

Line 3 (not illustrated) specifies that the proposition **wear_tux** stays false until m_{dock} turns true. Line 4 declare **rooms 1, 2** and **3** as a region group **dining_rooms**.

In Fig. 5.2b, the motion resulting from Line 5 is shown by the white curve. At point **C**, the sensor proposition $\pi_{customer}$ becomes true while π_{order} is still false, so the memory $m_{customer,order}$ becomes true, which reminds the robot to visit **dining_rooms**. At point **D**, $\pi_{customer}$ turns false but $m_{customer,order}$ stays true, and the robot keeps patrolling the **dining_rooms**, as represented by the dotted curve. At point **E**, π_{order} turns true,

which results in $m_{customer,order}$ going back to false, and so the patrolling requirement is forgotten. Line 6 is shown by the black curve: π_{order} and $m_{order,kitchen}$ become true, which causes the robot to move towards the **kitchen**. Observe that π_{order} is changed into false right away since the order is no longer observed, but this has no influence on the robot motion since the memory proposition $m_{order,kitchen}$ is still true. At point **F** in the **kitchen**, the memory $m_{order,kitchen}$ turns false, so that the robot is ready to wait on another **customer**.

CHAPTER 6

CONCLUSIONS

The thesis addresses two challenges of exploring unknown region and encoding implicit memory operation.

For the first challenge, this thesis introduces a new grammar for specifying high-level task over the potential undefined regions and describes the algorithms to automatically re-synthesize the controller when a new region is added to the current workspace. This includes modifying the discrete abstraction of the workspace, preserving the robot's state and history of task progress such that the robot resume the desired behavior. The proposed algorithms were successfully tested in experiment. The entire re-synthesis process is completely automatic and the generated controller is guaranteed to be correct. If the task can no longer be achieved in the modified workspace, the system will provide explanations for the failure.

For the second challenge, this thesis presents a structured English grammar for specifying high-level tasks that allow implicit memory operation. The grammar automatically generates extra propositions for recording events as well as LTL formulas for changing their values accordingly. Users are allowed to use the grammar for both robot and environment assumptions. The enriched grammar is shown to significantly reduce the length and increase the readability of the English specification compared to the original grammar.

BIBLIOGRAPHY

[1] Calin Belta and Luc C.G.J.M. Habets. Constructing decidable hybrid systems with velocity bounds. In *IEEE Conference on Decision and Control*, Bahamas, 2004.

[2] Calin Belta, Volkan Isler, and George J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

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

[4] S. R. K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *ACL*, pages 1268–1277, 2010.

[5] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[6] D.C. Conner, A.A. Rizzi, and H. Choset. Composition of Local Potential Functions for Global Robot Control and Navigation. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 3546 – 3551, Las Vegas, NV, October 2003.

[7] Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul W. Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *ICRA*, pages 4163–4168, 2009.

[8] 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.

[9] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control*, pages 2222–2229, 2009.

[10] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from LTL specifications. In *Hybrid Systems: Computation and Control*, volume 3927, pages 333–347, 2006.

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

[12] S. R. Lindemann and S. M. LaValle. Smooth feedback for car-like vehicles in polygonal environments. In *IEEE Conference on Robotics and Automation*, 2007.

[13] Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *HRI*, pages 251–258, 2010.

[14] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, Charleston, SC, Jenuary 2006.

[15] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[16] Vasumathi Raman and Hadas Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP. In *CAV*, pages 663–668, 2011.

[17] Vasumathi Raman, Bingxin Xu, and Hadas Kress-Gazit. Avoiding forgetfulness: Structured english specifications for high-level robot control with implicit memory. *International Conference on Intelligent Robots and Systems*, 2012.

[18] Shahar Sarid, Bingxin Xu, and Hadas Kress-Gazit. Guaranteeing high-level be-

haviors while exploring partially known maps. *Robotics: Science and Systems*, 2012.

[19] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Hybrid Systems*, pages 101–110, 2010.