Avoiding Forgetfulness: Structured English Specifications for High-Level Robot Control with Implicit Memory

Vasumathi Raman[1], Bingxin Xu and Hadas Kress-Gazit[2]

Abstract— **This paper addresses the challenge of incorporating event memory into the automatic synthesis of hybrid controllers for high-level reactive robot behavior. The goal is to provide a natural, concise grammar for specifying high-level tasks that require remembering past events, and to ensure that the required memory is correctly updated during controller execution. To this end, a structured English grammar for specifying high level behavior is provided 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 action sequences, etc. The proposed framework also guarantees the correctness of memory operations during continuous execution. The approach is implemented within the LTLMoP toolkit for reactive mission planning.**

I. Introduction

High-level robot control has recently seen the application of formal methods to automatically synthesize controllers that achieve the specified behavior. These high-level behaviors include reacting to environmental events at runtime, and temporally extended goals; such behaviors are often required for applications like search and rescue missions and autonomous driving. Most formal methods frameworks for generating verifiable high-level control rely on a discrete abstraction of the underlying system dynamics, use model checking (e.g. [1], [2], [3], [4]) or efficient synthesis techniques (e.g. [5], [6]) to generate control laws on this discrete model, and transform them into provably correct low-level robot controllers that achieve the specified behavior. The desired properties are usually expressed using a temporal logic, such as Linear Temporal Logic (LTL)[7].

Synthesizing controllers that satisfy reactive behaviors requires specifications that describe the robot's goals, as well as assumptions on the environment it operates in. The generated robot controllers are guaranteed to satisfy the desired specification in any environment that satisfies the modeled environment assumptions. One challenge that emerges is the specification of events to be remembered, whether external (i.e. perceived in the environment) or internal (i.e. executed by the robot). Since the synthesized controllers correspond to finite state machines, they are Markovian, in that the robot's actions depend entirely on the current state and the environment inputs. This requires that memory of past events be directly encoded in the state. Explicitly specifying memory operations in a high-level specification is not an ideal solution in two respects: 1) it shifts the burden of reasoning about memory requirements to the user; 2) even with explicitly defined memory management, the system may still fail to remember events during continuous execution of the synthesized discrete controller. These two problems are described in more detail with examples in Section III. This paper addresses the challenge of automatically managing memory operations given a specification that may not include explicit memory management.

There are several approaches to using language for controlling robots. These include frameworks for translating instructions in unconstrained natural language to formal goal descriptions and action scripts for robot navigation and manipulation [8], [9], [10], and approaches that map high-level instructions to sequences of commands that fill in the missing information [11]. While the language used in this paper is structured English rather than natural language, this is the first work to address the automatic integration of implicit memory operations when synthesizing a finite-state robot controller for a high-level specification. The memory management strategies described here are implemented within Linear Temporal Logic MissiOn Planning (LTLMoP)[12], [13], an open source, modular, Python-based toolkit that automatically generates and implements hybrid controllers corresponding to structured English specifications describing high-level robot behaviors, using the approach of [5]. The synthesized controllers can either be embedded within a simulator or used with physical robots. The most recent version of LTLMoP can be downloaded online[1].

The paper is structured as follows. Section II provides an overview of the controller synthesis and corresponding specification language. Section III describes the problems addressed by this paper. Section IV presents the proposed grammar for implicitly specifying memory operations, and Section V discusses measures to ensure correct memory updates during continuous execution. Section VI illustrates the robot behavior produced using the described approach for a simple specification that uses event memory. The paper concludes with a discussion of future work in Section VII.

II. Background

This section discusses the underlying logical formalism and specification language, an overview of the synthesis of

This work was supported by NSF CAREER CNS-0953365, ARO MURI (SUBTLE) W911NF-07-1-0216 and NSF ExCAPE

[1]V. Raman is with the Department of Computer Science, Cornell University, Ithaca, NY, USA vraman@cs.cornell.edu

[2]B. Xu and H. Kress-Gazit are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, USA {bx38, hadaskg}@cornell.edu

[1]http://ltlmop.github.com

provably-correct robot control from logical formulas, and a description of the Linear Temporal Logic MissiOn Planning (LTLMoP) toolkit [12].

A. Linear Temporal Logic (LTL)

The underlying logical formalism used to synthesize robot controllers in this work is Linear Temporal Logic (LTL). LTL is a modal logic that includes temporal operators, allowing formulas to specify the truth values of atomic propositions over time. LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following recursive grammar:

Syntax: Let AP be a set of atomic propositions. LTL formulae are constructed recursively from $\pi \in AP$ as follows:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \Box \varphi \mid \Diamond \varphi$$

where $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", $\Box$ is "always", and $\Diamond$ is "eventually" (the LTL "until" operator $\mathcal{U}$ is omitted for this work).

Semantics: The truth of an LTL formula is evaluated over infinite executions of a finite state machine representing the system: a state is an assignment of truth values to propositions in AP, and an execution is an infinite sequence of truth assignments to $\pi \in AP$. A finite state machine satisfies a formula if every execution satisfies it. For a formal definition of the semantics, the reader is referred to [7].

B. Discrete Abstraction

Applying formal methods to inherently continuous problems in robotics requires a discrete abstraction of the problem to enable description with a formal language. This work considers robot specifications using the GR(1) fragment of LTL [14]. The workspace is modeled as a two dimensional polygonal environment, and the possible motion of the robot in the workspace is abstracted using a graph where each node represents a region and edges represent transitions between adjacent regions.

For specifications on this discrete abstraction, the atomic propositions consist of a set $\mathcal{X} = \{x_1, ..., x_m\}$ of environment propositions corresponding to abstract sensor information (e.g. "object detected"), and a set $\mathcal{Y} = \{r_1, ...r_n, a_1, ..., a_k\}$ of robot propositions corresponding to the robot location (r_i is true when the robot is in region i) and actions a_j (e.g."raise the flag"). The fragment of LTL considered in this work follows [5], 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, while φ_s represents the desired robot behavior; specifications belong to a fragment of LTL that allows efficient synthesis [14].

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

Example 1 *Consider a robot waiting tables at a restaurant. At the beginning of each 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 and places the order with the chefs. The restaurant map is depicted in Fig. 4.*

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 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.

C. Structured English Task Specification in LTLMoP

LTLMoP [12] is a Python-based, open-source toolkit allowing users to control physical and simulated robots by specifying high-level instructions in structured English. To allow non-technical users to write robot specifications even if they are unfamiliar with LTL, LTLMoP includes a parser that automatically translates English sentences belonging to a defined grammar into LTL formulas [15], [16]; the grammar includes conditionals, goals, safety sentences and non-projective locative prepositions such as "between" and "near." Structured English circumvents the ambiguity and computational challenges associated with natural language, while still providing a more intuitive medium of interaction than conventional programming languages.

LTLMoP's structured English grammar allows the user to define desired robot behaviors (including reactive behaviors, e.g., if you see a truck, unload it) and specify assumptions about the behavior of the environment (e.g., a truck will never be seen in the kitchen) using a more natural formalism than the underlying LTL. There are two primary types of properties allowed in a specification – *safety* properties, which guarantee that "something bad never happens", and *liveness* conditions, which require that "something good (eventually) happens"; these correspond respectively to LTL formulas with $\Box$ (always) and $\Box\Diamond$ (always eventually). This paper enriches the structured English grammar by allowing specifications that express implicit memory of events, as described in Section IV.

D. Control Generation

Given a task specification and a description of the workspace topology, LTLMoP applies the efficient synthesis algorithm introduced in [14] to construct an implementing automaton (if one exists). If no such automaton exists, the

user is presented with information about the cause of the unsynthesizability [13], [17]. If an automaton is obtained, it is transformed into a hybrid controller which can be deployed on physical robots or in simulation. Details of the synthesis and the resulting hybrid controller can be found in [5], [12].

Conceptually, the process of transforming a high-level task described in structured English in LTLMoP to correct control inputs for a robot is composed of three stages:

1) Parsing the structured English sentences into an LTL formula defined over an abstraction of the problem.
2) Transforming the logic formula into an automaton.
3) Executing the automaton as a hybrid controller, where a transition between states corresponds to executing a set of low-level continuous controllers.

III. Problem Formulation

The task specification described in Example 1 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.

Consider the requirement, "Every time an order is made, go to the kitchen". In the structured English grammar defined in [15], the closest match to this specification is "If order then go to kitchen", which in turn corresponds to the LTL formula $\square\diamondsuit(\pi_{order} \implies \pi_{kitchen})$. However, the semantics of this LTL formula require that either $\pi_{kitchen}$ holds infinitely often or $\neg\pi_{order}$ holds infinitely often. If the robot senses π_{order} in a single time-step but not in any subsequent ones, the above liveness requirement is satisfied whether or not it goes to the kitchen. Therefore the synthesized automaton may not include any states in which the robot is in the kitchen. Moreover, even if the automaton does include a transition to the kitchen when an order is sensed, continuous execution may result in this order being forgotten. For example, 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 that it has to go to the kitchen.

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 propositions that are set on the associated event. Note that it is possible to modify the above specification using the original grammar in [15] to explicitly specify changes in memory by introducing a new proposition m_{order} (and the corresponding Structured English phrase "memo_order"). Listing 4 shows the additional structured English and corresponding LTL that accomplishes this.

Definition 1 (Memory Propositions) *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 above, m_{order} is a memory proposition whose value responds to π_{order}.

As shown above, the grammar introduced in [15] requires the user to explicitly introduce memory propositions and specify the events that cause them to be true. This places the burden of reasoning about what memory needs to be recorded on the user.

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

$$\square\diamondsuit(m_{order} \implies \pi_{kitchen})$$

Problem 1

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

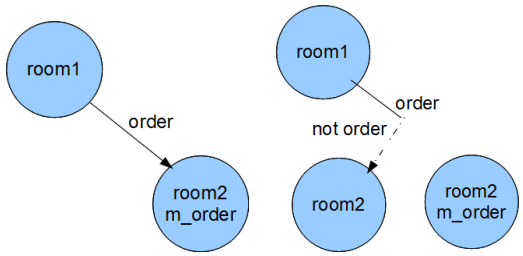
A second problem stems from the continuous execution of the synthesized controller described in Section II. Given a discrete transition between two states, the motion controller for driving the robot between regions is activated first. The remaining controllers for other propositions that change value over the transition are only activated (or deactivated) once the robot has entered the new region. Assuming the non-motion action controllers are instantaneous, this ensures that the discrete transition is safely executed. Consider what happens when the robot senses an order in $room_1$, while moving to $room_2$. The memory proposition m_{order} would not be set until after the robot has crossed into $room_2$. The transition from $room_1$ to $room_2$ is non-instantaneous, but as long as the robot is sensing an order until it crosses into $room_2$, the memory proposition will be set (as in Fig. 1(a)).

However, this continuous execution paradigm can lead to memory loss since the updating of memory propositions is delayed until the robot has moved to the new region. Consider what happens if the robot stops sensing the order when still in $room_1$ (Fig. 1(b)). The memory proposition m_{order} will never be set, and this order will be forgotten.

Problem 2 *Given a specification $\mathcal{S}$, the corresponding memory propositions $\mathcal{M}$ and LTL formulas specifying the memory operations Φ, modify Φ to ensure that every $m_\phi \in M$ is set in response to ϕ during continuous execution of the synthesized automaton.*

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

TABLE I: Syntax and Grammar



(a) Original transition (b) Forgetfulness

Fig. 1: Forgetfulness with continuous execution

IV. Structured English with Implicit Memory

This section describes the additional constructs added to the structured English grammar for automatically generating and using implicit memory propositions. The new grammar affords users the expressive power to imply the use of several kinds of memory propositions, based on which the robot is constrained to remember events in the history of both environment and robot behaviors. The presented constructs extend the grammar described in [15].

A. Structured English Grammar

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

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, $\square \diamond (\pi_{region})$ will drive the robot to visit the *region* infinitely often, while $\square((\bigcirc m\_r) \Leftrightarrow (m\_r \vee \bigcirc \pi_{region})) \wedge \square \diamond (m\_r)$ will not result in repeated 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 memory. The symbol Θ represents structured English phrases of the form,"you are activating *action*", "visit *region*", etc, that conform to the grammar described in [15]. 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 Types 2 and 3, the phrase "at least once" is optional if following "visit/go to", and "visit π_{region}" is translated into $\square((\bigcirc m\_r) \Leftrightarrow (m\_r \vee \bigcirc \pi_{region})) \wedge \square \diamond (m\_r)$ in contrast with the definition in [15]; 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 over sensor, action and region propositions. 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 [15], "do/activate/sense/in π" implies a safety requirement, and "repeatedly visit/infinitely often do π" implies a liveness.

B. 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 is met, 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 robot must fulfil a requirement each time the condition is true.

Finally, Type *5 propositions combine Types 1 and 3, and are introduced when directing the robot to fulfil a requirement at least once if the condition has been fulfiled.

Consider the task described in Example 1. Using the proposed (enriched) grammar, the specification is captured in Listing 2. This concise, intuitive specification illustrates the power of the proposed enriched structured English grammer. The memory propositions created by parsing the specification are: m_{check_in}, m_{truck}, m_{dock}, $m_{customer,order}$, $m_{order,kitchen}$.

Listing 2 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.
```

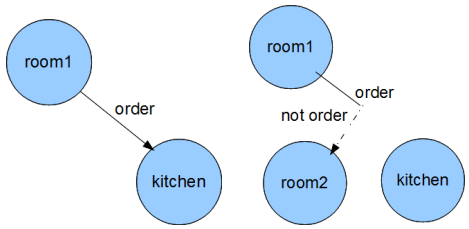
In comparison, the equivalent specification without implicit memory propositions in Listing 3 (using the grammar in [15]) is significantly longer (139 words vs. 56) and far less readable. In general, explicitly specifying memory storage is always less concise. For example, the single sentence in line 3 of Listing 2, "Only once you have visited the loading deck, then do wear_ tux from now on", requires the three sentences in Lines 4, 6 and 7 in Listing 3.

V. Guaranteeing Memory During Execution

Recall Problem 2, which is an outcome of the execution paradigm for the synthesized automaton. This problem can be solved by introducing a "stay there" condition, which forces the robot not to change location when changing memory (Fig. 3(a)). In the example in Listing 4, an LTL formula $\square(\neg m_{order} \wedge \bigcirc \pi_{order} \implies \bigcirc \pi_r \iff \pi_r)$ would be added to the specification for each room r. This "stay there" condition can be automatically added to the LTL

Listing 3 Original English specification equivalent to Listing 2

```
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
```



(a) Original transition (b) Forgetfulness

Fig. 2: Forgetfulness during self-resetting

specification whenever a memory proposition turns true; this avoids the situation described above in which an event is forgotten, because the memory proposition is set immediately after sensing an order.

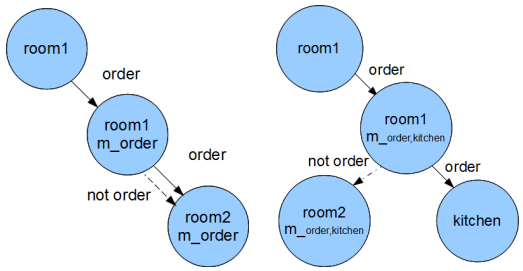
Consider the LTL specification in Listing 4, which uses the enriched grammar described in this paper. This specification prioritizes going to the kitchen over sensing an order – if both are true in the same time step, $m_{order,kitchen}$ is not set, as shown in Fig. 2(a). Fig. 3(b) shows that adding a "stay there" condition when $m_{order,kitchen}$ is set causes the self-resetting proposition $m_{order,kitchen}$ to be turned on and off over separate time steps. This again avoids the memory loss that occurs without the "stay there", depicted in Fig. 2(b), where $m_{order,kitchen}$ is never set if the motion to the kitchen is aborted, and the robot proceeds to patrol room 2 with no memory of the order.

Listing 4 Specification demonstrating the additional sentences required to avoid forgetfulness

After each time you have sensed order, visit kitchen
$\square(\bigcirc m_{order,kitchen} \iff ((\bigcirc \pi_{order} \vee m_{order,kitchen}) \wedge \neg \bigcirc \pi_{kitchen}))$
$\square\diamondsuit(m_{order,kitchen} \implies \pi_{kitchen})$

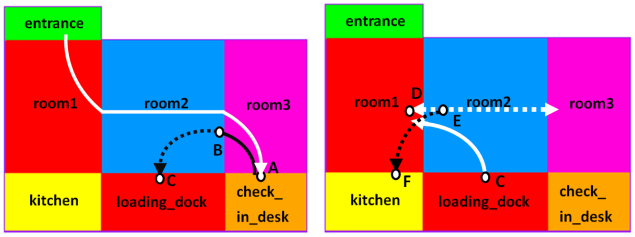
VI. Simulation of Synthesized Robot Behavior

This section demonstrates a simulation of the robot behavior synthesized in LTLMoP using the specification in Listing 2. The motion specified by Lines 1-2 of the specification



(a) Continuous Execution (Fig. 1) (b) Self-resetting (Fig. 2)

Fig. 3: Staying in place to avoid forgetfulness



(a) Lines 1 (white) and 2 (black) (b) Lines 5 (white) and 6 (black)

Fig. 4: Restaurant workspace and robot trajectories satisfying Listing 2. Points represent events that change the truth values of memory propositions, curves represent motion before (solid) and after (dashed) the events

is shown in Fig. 4(a), and that of Lines 5-6 in Fig. 4(b). Line 3 (not illustrated) specifies that the proposition **wear_tux** stays false until m_{dock} turns true. Line 4 declares **rooms 1, 2** and **3** as a region group **dining_rooms**.

In Fig. 4(a), Line 1 is shown by the white curve: the robot starts at the **entrance** and moves to the **check_in_desk**. At point **A**, the memory proposition m_{check_in} turns true and stays true. The black curve fulfils Line 2: at point **A**, the robot enters room 3 and sees a truck; π_{truck} and m_{truck} become true, and the robot moves through room 2 towards **loading_dock**. At **B**, π_{truck} turns false while m_{truck} stays true; this doesn't influence the robot motion, as shown by the dotted curve. At **C**, the memory proposition m_{dock} turns true and stays true.

In Fig. 4(b), an instance of 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, reminding the robot to visit **dining_rooms**. At **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 **E**, π_{order} turns true, which results in $m_{customer,order}$ going back to false, and so the patrolling requirement is erased from memory. π_{order} and $m_{order,kitchen}$ now become true, which causes the robot to move towards the **kitchen** to satisfy Line 6, as shown by the black curve. Observe that π_{order} is changed to 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**.

VII. Conclusions and Future Work

This paper presents a structured English grammar for specifying high-level tasks that require remembering past events. It includes examples illustrating the expressive power of this grammar, including a simulation of the resulting robot behavior. It also describes measures to ensure correctness of memory during the continuous execution of controllers synthesized from specifications in the described grammar. The enriched grammar is shown to reduce the length and complexity of the user-defined specification needed to produce the same behavior. Future work will further enrich the grammar to allow more complex memory operations, including remembering sequences of events. Another future direction is to improve the structured English grammar to bring it closer to natural language, while still maintaining an unambiguous interpretation for specifications.

References

[1] M.Kloetzer and C.Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transaction on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

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

[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*. IEEE, 2010, pp. 2689–2696.

[4] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. LaValle, "Controlling wild bodies using linear temporal logic," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[5] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Hybrid Systems*, 2010, pp. 101–110.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[8] J. Dzifcak, M. Scheutz, C. Baral, and P. 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*, 2009, pp. 4163–4168.

[9] C. Matuszek, D. Fox, and K. Koscher, "Following directions using statistical machine translation," in *HRI*, 2010, pp. 251–258.

[10] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *AAAI*, 2011.

[11] S. R. K. Branavan, L. S. Zettlemoyer, and R. Barzilay, "Reading between the lines: Learning to map high-level instructions to commands," in *ACL*, 2010, pp. 1268–1277.

[12] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2010, pp. 1988 – 1993.

[13] V. Raman and H. Kress-Gazit, "Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP," in *CAV*, 2011, pp. 663–668.

[14] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI 06*. Springer, 2006, pp. 364–380.

[15] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[16] H. Kress-Gazit and G. J. Pappas, "Automatic synthesis of robot controllers for tasks with locative prepositions," in *ICRA*, 2010, pp. 3215–3220.

[17] V. Raman and H. Kress-Gazit, "Automated feedback for unachievable high-level robot behaviors," in *ICRA*, 2012, pp. 5156–5162.