2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Open-World Mission Specification for Reactive Robots

Spyros Maniatopoulos, Matthew Blair, Cameron Finucane, and Hadas Kress-Gazit

***Abstract*— Recent advances have enabled the automatic generation of correct-by-construction robot controllers from high-level mission specifications. However, most current approaches operate under the closed-world assumption, i.e., only elements of the world explicitly modeled *a priori* can be taken into account during execution. In this paper, we tackle the problem of specifying and automatically updating the missions of robots operating in worlds that are open with respect to new elements, such as new objects and regions of interest. We demonstrate our approach in a scenario featuring a robotic courier whose world is open with respect to letters addressed to new recipients.**

I. INTRODUCTION

In order for robots to become widely accessible, users must be able to command them by specifying high-level behaviors, rather than programming low-level robot controllers. This type of robot control has recently become possible by applying tools and ideas from formal methods and hybrid systems to the field of robotics. The resulting methodologies translate high-level tasks to discrete and subsequently low-level continuous controllers in a correct-by-construction manner [1]–[7]. *Reactive* approaches, e.g. [2], [4], [7], account for a dynamic – and possibly adversarial – environment, and are thus of particular interest.

These approaches operate under the closed-world assumption. The robot only accounts for what was modeled before execution, and no information can be added autonomously, i.e. without a human re-specifying the mission. However, robots have moved from the cloistered assembly line to such highly unstructured workspaces as the Fukushima Daiichi nuclear disaster site [8]. In addition, robots are sometimes asked to incorporate new functions during execution. Examples include robots that learn on-the-fly [9] or query online knowledge repositories [10]. Finally, the world may be open with respect to goals added after the robot has been deployed, as in the case of urban search and rescue tasks [11], military scenarios [12], and autonomous space and planetary exploration missions [13].

Example 1: Consider a robotic courier, i.e., a "mailbot", (see Fig. 1) deployed in a school or company building. Its task is to collect letters and deliver them to the recipients' offices. This mission is relatively simple, and yet the robot's world is already open w.r.t. letters addressed to recipients about which the robot does not know.

In order to obtain controllers for open worlds, a number of challenges have to be overcome. First, the world can be open w.r.t. its spatial components. An aspect of this issue has been tackled in [14] and [15], where the authors introduced local resynthesis as a way to account for local topological changes in the robot's workspace. Additionally, an approach that addresses the unreactive equivalent of the same problem appeared in [16]. Furthermore, the aforementioned approaches deal with changes in the internal structure of the workspace. We are interested in expanding the workspace upon discovery of new regions. In addition, we are interested in augmenting the mission with additional objectives, a situation that may arise if the robot can discover new regions or objects of interest. The work in [17] allows the robot's workspace to expand, but does not account for the discovery of non-spatial aspects of the world, such as learning new actions or responding to unexpected events. Moreover, there is the the challenge of specifying open-world missions in the first place, since the world is only partially known prior to execution. Specification languages and abstractions have been developed for similar tasks with both formal [11], [18] and statistical [19] methods. This paper differs from previous approaches in its use of correct-by-construction controller synthesis in an open world, requiring new techniques.

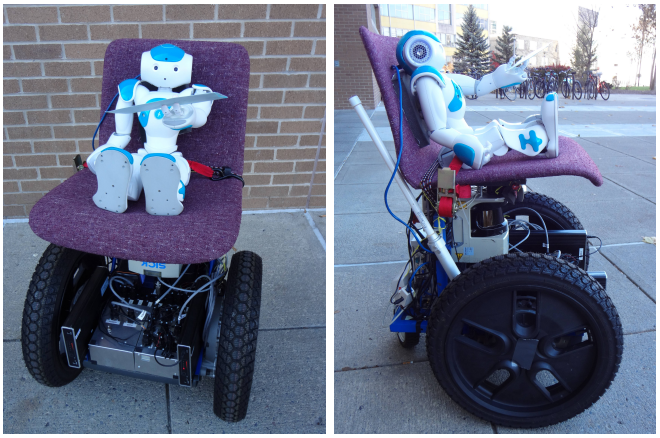


Fig. 1: Our robotic courier: an Aldebaran Nao humanoid robot is mounted on a Segway-based, mobility and sensing platform.

In this paper, we first propose *open-world abstractions*, which allow us to specify high-level robot behaviors in terms of elements in the world that are unknown prior to execution. Furthermore, they allow the specification to be updated in an autonomous fashion, as new elements of interest are discovered. The new elements include, but are not limited to, the sensing of new objects and events, and the discovery of new regions of the workspace, and they may lead to new mission constraints and/or goals. We show how the mission specification can be systematically and automatically rewritten on-the-fly, in order to correctly incorporate these

This work was supported by NSF CAREER CNS-0953365 and NSF ExCAPE.

The authors are with the Autonomous Systems Lab, Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA, {sm2296, meb286, cpf37, hadaskg}@cornell.edu

new elements of the open world. Finally, our high-level approach allows the user to control how the new elements will be accounted for, and when the robot should start operating according to the updated mission specification. Therefore, our work differs from, and is complementary to, [14] and [15], which focus on directly revising the robot's discrete controller to account for changes in the workspace.

The paper is organized as follows: Section II provides the necessary background on logic-based reactive mission planning. Section III formally states the problem we are addressing. The proposed abstractions, which we will use to specify tasks over open worlds, are defined in Section IV. Section V presents our approach to systematically rewriting the mission specification. Simulations of the mailbot scenario (Example 1), which is revisited throughout the paper, are provided in Section VI. Finally, our conclusions, as well as possible research directions, are summarized in Section VII.

II. PRELIMINARIES

A. Controller Synthesis

We present an overview of our reactive controller synthesis process. We refer the reader to [2] for a detailed presentation.

Discrete Abstraction: First, the robot's workspace is partitioned using a finite number of cells. The Boolean region propositions $\mathcal{R} = \{r_1, r_2, \ldots, r_{N_1}\}$ denote the location of the robot in the partition, and are thus mutually exclusive. In addition, the robot can perform actions $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_{N_2}\}$, which can involve the use of actuators, e.g. grasping an object, turning an on-board camera on, or the activation of memory propositions, e.g. having visited a region at least once. Region and action propositions together form the set of *robot* propositions $\mathcal{Y} = \mathcal{R} \cup \mathcal{A}$.

Furthermore, the robot has sensors, whose value, whether continuous or discrete, is abstracted by the Boolean *environment/sensor* propositions $\mathcal{X} = \{x_1, x_2, \ldots, x_{N_3}\}$. The sensor propositions allow the robot to react to its environment.

Linear Temporal Logic: We are interested in missions with complex requirements, such as surveillance, avoidance, and response to events. Thus, we use Linear Temporal Logic (LTL) formulas to specify desired high-level behaviors. LTL consists of propositions, and Boolean ($\neg, \wedge, \vee$) and temporal (next, $\bigcirc$, and until, $\mathcal{U}$) operators. Additional temporal operators, eventually ($\Diamond$) and always ($\Box$) can be derived. In our setting, LTL formulas are constructed from atomic propositions $\pi \in AP = \mathcal{X} \cup \mathcal{Y}$. The semantics of an LTL formula φ are defined over an infinite sequence σ of truth assignments to the atomic propositions. Let $t \in \mathbb{N}$ be the position index of σ. Then, $\sigma(t)$ denotes the subset of propositions $\pi \in AP$ that are `True` at position t of the sequence σ, and $\pi[t]$ denotes the valuation of a particular proposition at position t.

Reactive Synthesis: Specifically, we use the GR(1) fragment of LTL, because its synthesis is tractable [20]. GR(1) formulas φ have an assume-guarantee structure between the environment (e) and the robot (system, s):

$$\varphi = (\varphi_e \Rightarrow \varphi_s); \varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e; \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$$

where i denotes initial conditions, t safety assumptions/requirements, and g liveness assumptions/requirements (goals) for the environment and the robot, respectively.

If the formula φ is realizable, a discrete robot strategy $\mathcal{S}$ is computed by solving a two-player game between the robot and the environment (adversary) [20]. The strategy, which is guaranteed to be winning for the robot, is augmented with simple low-level controllers to form a hybrid controller [2].

B. LTL Mission Planning

The Linear Temporal Logic Mission Planning (LTLMoP) toolkit [21] is a Python-based, open-source toolkit for controlling physical and simulated robots using high-level behavior specifications.[1] LTLMoP allows users to specify missions in either pure LTL, natural language via a module called SLURP [22], or a formal grammar called Structured English [23]. Structured English is designed to enable users to intuitively understand behavior specifications and this paper will use it for demonstration. Specifications in Structured English are parsed using a feature-based context-free grammar and translated into formulas of LTL for controller synthesis. LTLMoP provides our framework for mission specification, performs reactive controller synthesis, and monitors execution of the resulting strategy.

III. PROBLEM STATEMENT

Using the terminology in Section II, we say that the world is *closed* if and only if AP, the set of atomic propositions, is fixed. That is, if the robot encounters a new element in its world (such as a new object), which was not modeled as a proposition $\pi \in AP$, it will essentially ignore it, even if it is relevant to the mission.

Three main reasons motivate the need for open-world mission specification and planning. First, in applications such as autonomous search and rescue scenarios, the mission is often specified before the robot has obtained full knowledge of the world. Second, the robot may have to incorporate new functions and respond to new events during execution. Finally, the mission's set of atomic propositions AP may be too large for the user to enumerate a priori.

We begin our formal analysis of the problem by defining a model of an open world. Let $\mathcal{W}$ be a sequence of sets of atomic propositions, $\mathcal{W} = AP_0 AP_1 \ldots AP_k$, where $AP_{k+1} \neq AP_k$, and $k \in \mathbb{N}$.

Also, let $\mathcal{D} = \{d_1, \ldots, d_m, \ldots, d_M\}$ represent the abstract sensors that are responsible for detecting M different *classes* of new elements in the open world, where $\mathcal{D} \subseteq \mathcal{X}_k \subset AP_k, \forall k$. For instance, one sensor d may detect new letters, whereas another new offices. The number of new elements that can be detected is not bounded by M, since each sensor in $\mathcal{D}$ can trigger multiple times over the course of a mission. In addition, let D be a function that reads the sensors abstracted by $\mathcal{D}$, and returns a new proposition if

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

the sensors detected an unmodeled element of the world:

$$D(m,t) = \begin{cases} \{\pi_{mt}\}, & \text{if } d_m[t] = \texttt{True} \\ \emptyset, & \text{if } d_m[t] = \texttt{False} \end{cases} \quad (1)$$

where $d_m[t]$ denotes the value of the binary sensor d_m at position t of the infinite sequence σ (Section II-A), and $\pi_{mt} \notin AP_t$. The new proposition π_{mt} can be either a sensor, an action, or a region proposition.

Putting everything together, we state the open-world model that we consider in this paper.

Definition 1 (Expanding Open World Model): The sequence $\mathcal{W}$ is an expanding open world model if the set of atomic propositions AP is updated as follows:

$$AP_{t+1} = AP_t \cup \bigcup_{m=1}^{M} D(m,t), \quad (2)$$

and if AP changes, AP_{t+1} is appended to $\mathcal{W}$:

$$\mathcal{W} = \begin{cases} \mathcal{W} \circ AP_{t+1}, & \text{if } \bigvee_{m=1}^{M} (d_m[t] = \texttt{True}) \\ \mathcal{W}, & \text{otherwise} \end{cases} \quad (3)$$

As a corollary, if the world is closed, then $\mathcal{W} = AP_0, \forall t$.

We will return to this definition in Section V, where we augment the update of AP_t with additional new propositions.

Notice the assumptions regarding $\mathcal{W}$ that Eq. (2) makes:

Assumption 1: $\mathcal{W}$ only models the *addition* of propositions. That is, $AP_k \subset AP_{k+1}$. In other words, propositions cannot be removed. Hence, $\mathcal{W}$ is dubbed an *expanding* open world model.

Assumption 2: $\mathcal{W}$ models a world that is open only w.r.t. new elements of M different types, i.e., those elements that can be sensed by the sensors $d_m \in \mathcal{D}$, $m \in \{1, \ldots, M\}$.

Assumption 1 is not limiting, since a proposition can be in AP without being part of the robot's controller. Furthermore, the value of a proposition can be restricted by adding safety requirements to $\varphi_t^{e,s}$, if necessary. The disadvantage of not removing propositions that are no longer pertinent to the mission is that the synthesis algorithm [20] will be searching for a winning strategy over a larger graph, which leads to a higher computation time. Assumption 2 limits the new events to which the robot will react, to those that it can sense and that are relevant to the mission. For instance, the mailbot (Example 1) should update its mission if it senses a new recipient, but not if someone tries to hand it a trash bag.

In order to obtain a mission specification and a robot plan that react to unmodeled elements of the world, we need to address two main problems. First, we should allow specifications that include operators outside of LTL, in order to enable the mission to adapt to $\mathcal{W}$. Then, we need to define the specific mechanism that will allow new propositions π_{mt} to be incorporated in the mission's LTL formulas.

The problem of rewriting a specification to allow for adding new propositions is stated below:

Problem 1 (Mission Specification Update): Let Λ be a specification language and $\mathcal{P}_\Lambda$ a parser for it. Augment Λ and $\mathcal{P}_\Lambda$, such that given a mission specification written in Λ, $\mathcal{M}_k = \mathcal{M}(AP_k) \in \Lambda$, a GR(1) formula $\varphi[k]$, and the latest set of atomic propositions, $AP_{k+1} \supset AP_k$, the mission $\mathcal{M}_k$ and formula φ_k are automatically updated as follows:

$$\mathcal{M}_{k+1} = \mathcal{M}(AP_{k+1}), \quad (4a)$$

$$\varphi[k+1] = \mathcal{P}_\Lambda(\mathcal{M}_{k+1}), \quad (4b)$$

where the initial conditions of the GR(1) formula $\varphi[k+1]$ are set such that $\sigma(t^+) \models \varphi_i^e[k+1] \wedge \varphi_i^s[k+1]$, where $\sigma(t^+)$ are the current truth assignments to the atomic propositions $\pi \in AP_{k+1}$ right after AP_k is updated.

These restrictions on $\varphi[k+1]$ should ensure that its initial conditions are compatible with the current environment and robot state. The initial valuations of the new propositions $\pi \in AP_{k+1} \backslash AP_k$ are set according to the sensor readings and the constraints in $\mathcal{M}_{k+1}$. Notice that the specification language Λ and the parser $\mathcal{P}_\Lambda$ only provide the semantics and mechanics for updating $\varphi[k]$. The resulting formula, $\varphi[k+1]$, depends on the user-defined mission specification $\mathcal{M}$.

Given the problem statement above, we first augment the specification language Λ with *open world abstractions* in Section IV. These abstractions allow us to specify tasks without explicitly referring to individual propositions. In Section V we show how an additional mechanism of Λ leverages the expressive power of open world abstractions to enable the systematic addition of new propositions to AP_k, and consequently the rewriting of $\mathcal{M}_k$ and $\varphi[k]$.

IV. OPEN-WORLD ABSTRACTIONS

A. Groups of Propositions

In order to write meaningful specifications over propositions without referring to them explicitly, we borrow notions of first-order logic to augment Λ. First-order logic differs from propositional logic and LTL by its use of predicates and quantification over entities, of which our approach is only concerned with quantification. Two types of quantification are allowed in first-order logic: universal quantification using $\forall$ and existential quantification using $\exists$.

Λ allows quantification over sets of propositions, which we refer to as groups. This allows users to specify reactive behaviors in terms of groups of propositions instead of explicitly naming propositions. Quantification behaves identically towards all propositions within a group, therefore they must all be grammatically interchangeable within a specification. This interchangeability is a property of propositions of the same type (i.e. sensors, action propositions, or regions), therefore we further constrain groups to be sets of propositions that are all of a single type. In summary, a group G has the following property: $G \subseteq \mathcal{X}_t \,|\, G \subseteq \mathcal{A}_t \,|\, G \subseteq \mathcal{R}_t$. This generalizes the work in [17], which focused only on groups of region propositions.

B. Group Quantifiers

For an expression in Λ to be quantified over a group of propositions, the expression must have both a reference to the group (e.g. the name of the group) and a quantifier. Λ is augmented with three quantifiers: 'any', 'all', and 'each'. For quantification to be meaningful in a robot's behavior

the parser $\mathcal{P}_\Lambda$ must be able to translate the semantics of these first-order logic operations into LTL formulas φ. In this section we define LTL interpretations of quantification.

Let $\bar{\varphi}$ be the translation of an expression in Λ into an expression containing only LTL operators and quantified groups. Using G to denote a group, we can write the quantification of groups in $\bar{\varphi}$ as 'any(G)', 'all(G)', or 'each(G)'. We will also use $[y/x]\bar{\varphi}$ to denote the expression that results from replacing all occurrences of x in $\bar{\varphi}$ with y.

The quantifier 'any(G)' in a sentence is translated as the logical disjunction of every proposition in the group G. That is, $\bar{\varphi}$ is translated into: $[p/\text{any}(G)]\bar{\varphi}$, where: $p = \bigvee_{\pi_i \in G} \pi_i$.

The quantifier 'all(G)' in a sentence is translated as the logical conjunction of every proposition in the group G. $\bar{\varphi}$ is therefore translated into: $[p/\text{all}(G)]\bar{\varphi}$, where: $p = \bigwedge_{\pi_i \in G} \pi_i$.

The quantifier 'each(G)' is similar to 'all(G)', but acts with a different semantic scope. 'each(G)' denotes that the entire expression is true for each of the propositions in G separately. Thus, $\bar{\varphi}$ is translated into: $\bigwedge_{\pi_i \in G} [\pi_i/\text{each}(G)]\bar{\varphi}$

Example 2: Consider a scenario where a robot is helping to manage a hotel. The robot has a group of regions representing the rooms in the hotel, a group of sensors indicating whether rooms are occupied, with a proposition for every room, and actions for cleaning a room, welcoming guests, and apologizing to guests. Part of the robot's specification might be expressed in English as: "If some rooms are occupied but not all rooms are occupied then welcome guests. If all rooms are occupied then apologize to guests. If none of the rooms are occupied then go to each room and clean the room." Using the quantifiers in Λ, this behavior can be expressed without explicitly naming individual propositions.

C. Proposition Correspondence

One consequence of writing specifications without individual propositions is that relationships between propositions become more difficult to denote in specifications. For instance if a robot were operating in a workspace with groups of regions that had one-to-one relationships, such as pick-up and drop-off locations, those relationships cannot be specified without explicitly naming individual propositions. To address this difficulty, our new Λ must allow the definition of correspondences between propositions in groups. A correspondence is a mapping from individual propositions to sets of propositions that contain at most one proposition from each group defined in the specification. Λ allows users to define correspondence either by explicitly mapping atomic propositions to one another or by mapping one group onto another. Correspondence is denoted by the function $\mathcal{C}_\pi : AP \rightarrow 2^{AP}$, which takes in an atomic proposition and returns the set of propositions that it has been defined to correspond to, where 2^{AP} denotes the power set of AP. We say that a proposition π_i corresponds to π_j if and only if $\pi_j \in \mathcal{C}_\pi(\pi_i)$, where $i \neq j$. For defining correspondences between groups, we define another correspondence function $\mathcal{C}_G$, which operates on groups of propositions. We say that a group G_i corresponds to G_j, written as $G_j \in \mathcal{C}_G(G_i)$, where $i \neq j$ if and only if the following two conditions hold:

$$\forall \pi_j \in G_j \exists! \pi_i \in G_i : \pi_j \in \mathcal{C}_\pi(\pi_i) \tag{5a}$$

$$\forall \pi_i \in G_i : \mathcal{C}_\pi(\pi_i) \cap G_j \neq \emptyset \tag{5b}$$

Where $\exists!$ denotes unique existence. Note that these conditions necessitate that $|G_i| = |G_j|$. In the remainder of this paper we will omit the subscripts on $\mathcal{C}$ when its usage is clear.

Once correspondence has been defined in a specification, Λ allows the use of a 'corresponding' operator to implicitly refer to the relationship between individual propositions without explicitly referring to the propositions themselves. To use our previous notation of $\bar{\varphi}$ and G, the 'corresponding' operator can be written as 'corresponding(G)'. The 'corresponding' operator is different from the quantifiers described earlier because its semantics depend on the presence of another quantified group in the same sentence. We can demonstrate this dependency with two English sentences: "Go to the corresponding rooms" and "Take all of the guests to their corresponding rooms". The first sentence is unclear because it is lacking the index of correspondence provided in the second sentence, namely "all of the guests". Therefore a sentence using correspondence has two relevant groups: G_q, the quantified group, and G_c, the 'corresponding' group. G_q may be paired with any of the available quantifiers, but the interpretation of quantified sentences using correspondence always results in conjunction over statements, as occurs with the 'each' quantifier. We can now express the translation of $\bar{\varphi}$ for a sentence using correspondence as:

$$\bigwedge_{\pi_i \in G_q} [\pi_i/q(G_q)][\mathcal{C}(\pi_i) \cap G_c/\text{corresponding}(G_c)]\bar{\varphi}, \tag{6}$$

where q is any of the available quantifiers. Note from our definition of correspondence that the set intersection in (6) results in at most one element. If the intersection is empty, meaning that a correspondence has not been properly defined, then the sentence produces a syntax error.

Example 3: Correspondence is useful in a scenario in which a robot performs an identical task over many sets of related propositions, such as waiting on tables at a restaurant. Our specification can define a group of region propositions representing the tables in the restaurant and a group of sensor propositions representing the robot being signalled by specific tables. We can specify our desired behavior in English as follows: "Calls correspond to tables. If you are sensing any calls then go to the corresponding table." The abstraction of propositions into groups makes the specification more compact and readable, less error-prone, and more easily extensible. To specify this behavior over another pair of call and table propositions, we need only add the new propositions to their appropriate groups.

V. OPEN-WORLD MISSION SPECIFICATION

This section describes our approach to specifying and updating robot missions taking place in open worlds. Our presentation takes the form of requirements on a specification

language Λ, such that mission specifications $\mathcal{M}(AP) \in \Lambda$, parsed by $\mathcal{P}_\Lambda$ satisfy Eqs. (4a) and (4b). First, we leverage the expressive power of open-world abstractions to specify high-level behaviors without referring to individual propositions $\pi \in AP_k$. Then, we show that the addition of new propositions to a mission specification $\mathcal{M}_k = \mathcal{M}(AP_k)$ has to follow certain rules, such that correspondence $\mathcal{C}_G$ between groups of propositions is maintained. Finally, we provide a mechanism that allows the user to explicitly specify when the updated specification $\mathcal{M}_{k+1} = \mathcal{M}(AP_{k+1})$, and consequently the GR(1) formula $\varphi[k+1] = \mathcal{P}_\Lambda(\mathcal{M}_{k+1})$, should be synthesized into a revised robot controller.

A. Defining Tasks over Open Worlds

The first requirement on specifications $\mathcal{M}_0 = \mathcal{M}(AP_0) \in \Lambda$ is the definition of groups of propositions G_i and the definition of correspondence, $\mathcal{C}_G$, between groups of propositions. Then, tasks that refer to individual propositions $\pi \in G_i$ can be replaced by tasks over G_i and $G_j \in \mathcal{C}(G_i),\ j \neq i$.

An advantage of open-world abstractions is the ability to specify a mission even if the groups contain no propositions prior to execution, i.e., $G_i = \emptyset$ in $\mathcal{M}_0$. In this case, the groups will be populated on-the-fly as new elements in the world are discovered (see Section V-B, and Example 5).

Example 4: We revisit Example 1 and make use of the open world abstractions. For illustrative purposes, we use Structured English [23] as the specification language Λ.

Mission specification: (partial) Autonomous Mailbot

```
1: Group Letters is letter1, letter2
2: Group Offices is office1, office2
3: Letters correspond to Offices
4: If you are sensing any Letter then
go to the corresponding Office
```

Let G_1 be `Letters` and G_2 be `Offices`. Line 3 establishes correspondence from G_1 to G_2, i.e., $G_2 \in \mathcal{C}(G_1)$. We use the notation $G_2 \in \mathcal{C}(G_1)$, rather than $G_2 = \mathcal{C}(G_1)$, because G_1 could potentially correspond to more groups, e.g. $\mathcal{C}(G_1) = \{G_2, G_3, \ldots\}$. Notice how the task in line 4 does not refer to any particular letters or offices. As already mentioned, even if $G_1, G_2 \in \mathcal{M}_0$ were empty, the user would still be able to specify the mission above.

Let a letter corresponding to a new recipient be sensed by a sensor $d_m \in \mathcal{D}$, e.g. labeled `newLetter`. Then, a new proposition, π_{mt}, e.g. labeled `letter3`, would be added to AP_t. There are two requirements for the specification $\mathcal{M}_{k+1}$ to reflect the addition of the new proposition. First, π_{mt} has to also be added to the group G_1. Second, an office proposition, $\pi' =$ `office3`, has to be created, if necessary, and added to G_2, in order to re-establish correspondence, i.e., $G_2 \in \mathcal{C}(G_1)$, where now $G_1' = G_1 \cup \{\pi_{mt}\}$, $G_2' = G_2 \cup \{\pi'\}$. Then, we would have $\pi' \in \mathcal{C}(\pi_{mt})$. These two requirements are generalized in Problem 2, a subset of of Problem 1.

Problem 2 (New Propositions and Correspondence): Given a mission specification $\mathcal{M}_k$, containing n groups of propositions, $\mathbb{G}_M = \{G_1, G_2, \ldots, G_n\}$, and a new proposition $\pi_{mt} = D(m,t)$, define a mechanism for (i) adding π_{mt} to any number of groups $\mathbb{G} \subseteq \mathbb{G}_M$, and for (ii) creating additional propositions, π^{ij}, and adding them to the groups that each $G_i \in \mathbb{G}$ corresponds to, i.e., to

$$\mathbb{G}_\mathcal{C} = \bigcup_{G_i \in \mathbb{G}} \mathcal{C}_G(G_i)$$

B. Rewriting the Mission Specification

To address Problem 2, we introduce another element to the specification language Λ, the *add to Group* operator.

Definition 2 (add to Group): The *add to Group* operator takes the new proposition $\pi_{mt} = D(m,t)$, and adds it to one or more groups of propositions, $\mathbb{G} \subseteq \mathbb{G}_M$. Furthermore, it adds additional propositions, π^{ij}, to the corresponding groups, $\mathbb{G}_\mathcal{C}$. That is, $(\mathbb{G}', \mathbb{G}'_\mathcal{C}) =$ `add_to`$(\pi_{mt}, \mathbb{G})$, where

$$\mathbb{G}' = \{G_i' \mid G_i' = G_i \cup \{\pi_{mt}\}, G_i \in \mathbb{G}\},$$
$$\mathbb{G}'_\mathcal{C} = \left\{{G_j}' \,\middle|\, {G_j}' = G_j \cup \left\{\pi^{ij}\right\}, G_j \in \mathcal{C}(G_i), G_i \in \mathbb{G}\right\}.$$

Returning to Example 4, the add to Group operator would be `add_to`$(\pi_{mt}, \{G_1\}) = (\{G_1\}, \{G_2\})$, where G_1 and G_2 stand for `Letters` and `Offices`, respectively. The equivalent Structured English task would be: `if you are sensing newLetter then add to group Letters`, where `newLetter` is the label of a sensor $d_m \in \mathcal{D}$. The new propositions are named by either a predefined naming scheme, prompting the human user, or using information from the low-level sensor that is abstracted by the proposition d_m. For example, the new "letter" proposition could be named after the recipient's name, e.g. `letter_johnDoe`, while the "office" might use the address printed on the letter, e.g. `Room_155`.

Since we are now adding propositions besides those added by the detection sensors $d_m \in \mathcal{D}$, as a direct result of the open world, we have to revise Definition 1 as follows:

Definition 3 (Expanding Open World Model): Our model $\mathcal{W} = AP_0 AP_1 \ldots AP_k$ is now updated such that the propositions added in order to maintain correspondence are also added to the set of atomic propositions. That is,

$$AP_{t+1} = AP_t \cup \bigcup_{m=1}^{M} D(m,t) \cup \bigcup_{m=1}^{M} \bigcup_{\pi \in D(m,t)} \mathcal{C}_\pi(\pi) \quad (8)$$

replaces Eq. (2). Eq. (3) remains the same.

Assumption 3: The *grounding* [24] of new propositions $\pi \in AP_{k+1} \backslash AP_k$ to the robot's physical world is obtained either from the robot's low-level sensors, which propositions $d \in \mathcal{D}$ abstract, or by prompting the human user (Section VI). Some auxiliary propositions, such as robot memory propositions, do not require grounding to the physical world.

For instance, in Example 4, a new letter would result in new propositions `letter3` and `office3`. The sensor proposition `letter3` would be grounded to letters bearing the new recipient's name. The information for grounding `office3` to a location on the robot's workspace can be inferred from the address on the letter, which the robot can look up on a map. If the new office is not part of the current map, the robot could prompt the user for its location, and then incorporate it as a new region proposition.

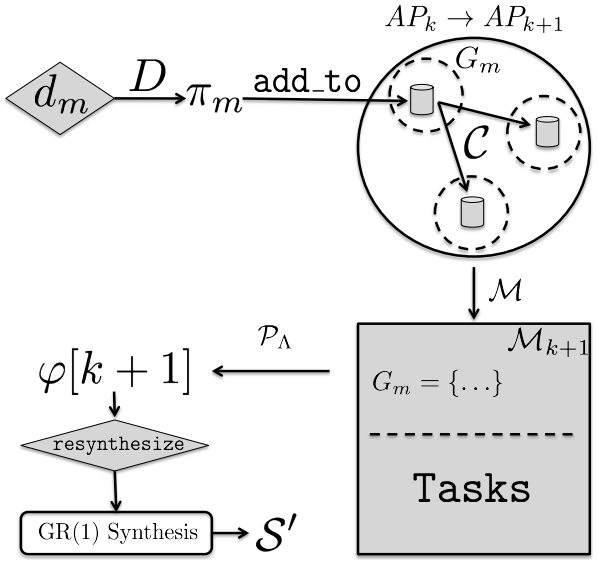


Fig. 2: Illustration of our approach. For clarity, a single new proposition π_m is added to a single group G_m, which has two corresponding groups, $\mathcal{C}(G_m)$. The `Tasks` part of $\mathcal{M}_{k+1}$ is immutable throughout the rewriting and resynthesis processes. That is, only the contents of groups G_m and $\mathcal{C}(G_m)$ change, not the tasks' description. Finally, $\mathcal{S}'$ denotes the revised robot controller.

C. Updating the Controller

The specification rewriting described so far has resulted in mission specification $\mathcal{M}_{k+1}$ that can be parsed to an LTL specification $\varphi[k+1]$, according to Eq. (4b). However, for the updates to be reflected in the robot's controller, $\varphi[k+1]$ has to be passed to the synthesis algorithm in order to extract a new winning strategy. This process is dubbed "resynthesis", and it is based on the work in [17].

Since we want to enable the user to specify when resynthesis should take place, we associate a robot proposition, e.g. `resynthesize`, with it. For example, in the setting of Example 4, and using Structured English, we could say: `if you are sensing newLetter then add to group Letters and resynthesize`. In this task, the robot would update its strategy immediately after detecting a letter for a new recipient. In a search and rescue setting, it might make more sense for the robot to finish executing other, higher priority tasks before resynthesizing. An excerpt from such a mission could be: `resynthesize if and only if you are not activating rescueSurvivor and you are activating resynthesisPending`.

The definition of the *add to Group* operator and *resynthesis* conclude the augmentation of the specification language Λ. Our approach is graphically summarized in Fig. 2.

VI. SIMULATION IN LTLMOP

We implemented the proposed approach to open-world mission specification in LTLMoP (see Section II-B). We augmented Structured English with *open-world abstractions* (groups, quantifiers, and correspondences), the *add to group* operator, and a *resynthesis* action.

Example 5: We revisit the mailbot scenario from Examples 1 and 4, and present the full mission specification, $\mathcal{M}_0$. The robot's workspace can be seen in Fig. 3a.

Initially, the robot has no information about specific letters or their recipients. Therefore, it patrols the regions in `PatrolRooms`. If it senses a new letter (see Fig. 3a), it will add a proposition to the group `Letters`. In simulation, LTLMoP prompts the user for the name of the proposition, e.g. `letter1`. Additionally, the *add to* operator has to add a proposition to the group `Offices`, in order to maintain the correspondence. LTLMoP prompts the user a second time, and the user inputs one of the region names, `r1`–`r6` (see Fig. 3b). The two user prompts simulate the process of scanning the letter for the recipient's name and address, in order to name and ground the new propositions (see Assumption 3).

Mission specification: Autonomous Mailbot

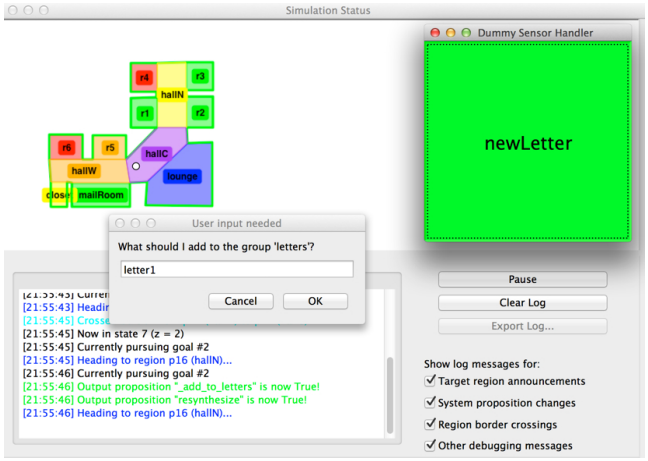
Group declarations:
```
Group Letters is empty
Group Offices is empty
Group PatrolRooms is mailRoom, hallW, hallN
```
Correspondence definitions: (immutable)
```
Letters correspond to Offices
```
Mission tasks: (immutable)
```
If you are sensing any Letters then go to the
corresponding Office
If you are not sensing any Letters then visit
each PatrolRoom
```
Open–World settings: (immutable)
```
If you are sensing newLetter then add to
group Letters and resynthesize
```

After the specification is rewritten, the LTL formula $\varphi[k]$ changes to $\varphi[k+1]$, according to Eqs. (4a) and (4b). The change involves adding a new mission goal: the delivery of `letter1` to the corresponding office `r1`. Specifically, compared to $\varphi_g^s[k]$, $\varphi_g^s[k+1]$ contains the additional liveness requirement: $\square\diamondsuit(\texttt{letter1} \Rightarrow \texttt{r1})$.

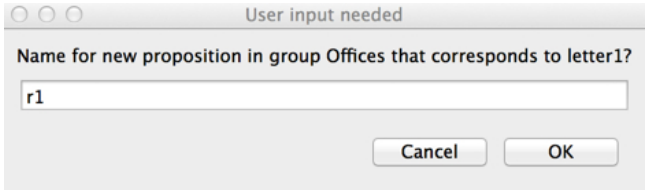
After resynthesis (see Fig. 3c), the robot can now sense letters in two ways. First, it could detect a letter addressed to the same recipient as `letter1`. In this case, `letter1` will become `True`, and the robot will deliver the letter to `r1`. Second, it could detect a letter addressed to a new recipient. In that case, the sensor `newLetter` will become `True`, and the *add to group* operator will once again cause new propositions to be added to `Letters` and `Offices`.

VII. CONCLUSIONS AND FUTURE WORK

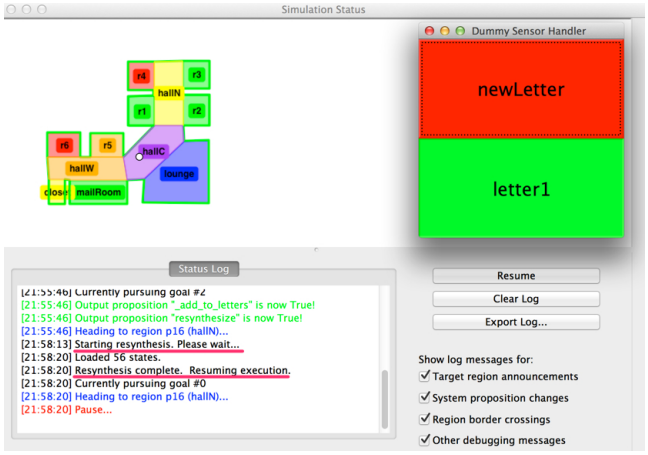
In this paper, we presented an approach to specifying and updating robot missions that take place in worlds open with respect to new elements, such as new objects and regions of interest. During execution, the new elements are translated into new propositions, which are automatically incorporated into the mission specification. This is possible via (i) *open-world abstractions*, which enable us to specify high-level behaviors without explicitly referring to individual propositions, and (ii) the *add to Group* mechanism, which systematically augments sets of propositions with new elements. A notable advantage of our approach is that it allows the user to specify how the new elements will be incorporated



(a) A new letter is detected. The user names the new proposition.



(b) The user grounds the corresponding office proposition to `r1`.



(c) `letter1` and `r1` were added to `Letters` and `Offices`.

Fig. 3: Simulation of Example 5 in LTLMoP.

into the mission, and under which conditions the updates will be reflected in the robot's controller.

A future research direction is the generalization of our model in order to also account for the removal of propositions that are no longer pertinent to the mission. In addition, our method of translating an updated specification to a new execution strategy is that of global resynthesis. We will investigate *local* resynthesis approaches, in the direction of [14], [15]. The ability to remove propositions, coupled with the efficiency of local resynthesis will allow us to deal with large-scale open worlds without synthesis becoming a computational bottleneck. Furthermore, we intend to generalize and formalize the introduction of elements of first-order logic to our specification language. On the implementation side, we are interested in a human-robot dialogue interface that would allow users to specify new tasks during execution.

References

[1] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion: State of the art and grand challenges," *Robotics and Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.

[2] 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.

[3] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *CDC*, 2009.

[4] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proc. of the 13th Int'l Conf. on Hybrid Systems: Computation and Control*, 2010.

[5] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. LaValle, "Controlling wild bodies using linear temporal logic," in *RSS*, Los Angeles, CA, USA, June 2011.

[6] A. Bhatia, M. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 55 –64, sept. 2011.

[7] A. Ulusoy, M. Marrazzo, and C. Belta, "Receding horizon control in dynamic environments from temporal logic specifications," in *Robotics: Science and Systems*, 2013.

[8] K. Nagatani, S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada, "Redesign of rescue mobile robot quince," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, 2011, pp. 13–18.

[9] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, "Learning to place new objects in a scene," *International Journal of Robotics Research*, vol. 31, no. 9, 2012.

[10] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The RoboEarth cloud engine," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany*, 2013, pp. 438–444.

[11] K. Talamadupula, J. Benton, P. Schermerhorn, S. Kambhampati, and M. Scheutz, "Integrating a closed world planner with an open world robot: A case study," *AAAI*, 2010.

[12] M. Klenk, M. Molineaux, and D. W. Aha, "Goal-driven autonomy for responding to unexpected events in strategy simulations," *Computational Intelligence*, vol. 29, no. 2, pp. 187–206, 2013.

[13] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, "Autonomous and autonomic systems: a paradigm for future space exploration missions," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 36, no. 3, pp. 279–291, 2006.

[14] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *IEEE Int'l. Conf. on Robotics and Automation*, 2012.

[15] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray, "Patching task-level robot controllers based on a local μ-calculus formula," in *IEEE Int'l. Conf. on Robotics and Automation*, 2013.

[16] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *IEEE Int'l. Conf. on Robotics and Automation*, 2013.

[17] S. Sarid, B. Xu, and H. Kress-Gazit, "Guaranteeing high-level behaviors while exploring partially known maps," in *Robotics: Science and Systems*, 2012.

[18] S. Joshi, P. Schermerhorn, R. Khardon, and M. Scheutz, "Abstract planning for reactive robots," *2012 IEEE International Conference on Robotics and Automation*, pp. 4379–4384, May 2012.

[19] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. Teller, and N. Roy, "Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation," *AAAI*, 2011.

[20] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *VMCAI*, Charleston, SC, January 2006, pp. 364–380.

[21] 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*, Oct. 2010, pp. 1988–1993.

[22] V. Raman, C. Lignos, C. Finucane, K. C. T. Lee, M. Marcus, and H. Kress-Gazit, "Sorry Dave, I'm Afraid I Can't Do That: Explaining Unachievable Robot Tasks Using Natural Language," *Robotics: Science and Systems*, 2013.

[23] 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.

[24] S. Coradeschi, A. Loutfi, and B. Wrede, "A short review of symbol grounding in robotic and intelligent systems," *KI*, vol. 27, no. 2, pp. 129–136, 2013.