# From Structured English to Robot Motion *

Hadas Kress-Gazit, Georgios E. Fainekos and George J. Pappas
GRASP Laboratory, University of Pennsylvania
Philadelphia, PA 19104, USA
{hadaskg,fainekos,pappasg}@grasp.upenn.edu

***Abstract*— Recently, Linear Temporal Logic (LTL) has been successfully applied to high-level task and motion planning problems for mobile robots. One of the main attributes of LTL is its close relationship with fragments of natural language. In this paper, we take the first steps toward building a natural language interface for LTL planning methods with mobile robots as the application domain. For this purpose, we built a structured English language which maps directly to a fragment of LTL.**

## I. INTRODUCTION

Successful paradigms for task and motion planning for robots require the verifiable composition of high level planning with low level controllers that take into account the dynamics of the system. Most research up to now has targeted either high level discrete planning or low level controller design that handles complicated robot dynamics (for an overview see [2], [16]). Recent advances [1], [4], [6], [17] try to bridge the gap between the two distinct approaches by imposing a level of discretization and taking into account the dynamics of the robot.

The aforementioned approaches in motion planning can incorporate at the highest level any discrete planning methodology [2], [16]. One such framework, is based on automata theory where the specification language is the so-called Linear Temporal Logic (LTL) [3]. In the case of known and static environments, LTL planning has been successfully employed for the non-reactive path planning problem of a single robot [8], [9] or even robotic swarms [12]. For robots operating in the real world, one would like them to act according to the state of the environment, as they sense it, in a reactive way. In our recent work [14], we have shifted to a framework that solves the planning problem for a fragment of LTL [21], but now it can handle and react to sensory information from the environment.

One of the main advantages of using this logic as a specification language is that LTL has a structural resemblance to natural language[1]. Nevertheless LTL is a mathematical formalism which requires expert knowledge of the subject if one seeks to tame its full expressive power and avoid mistakes. This is even more imperative in the case of the fragment of Linear Temporal Logic that we consider in this paper. This fragment has an assume-guarantee structure that makes it difficult for the non-expert user even to understand a specification, let alone formulate one.

*This work is partially supported by National Science Foundation EHS 0311123, National Science Foundation ITR 0324977, and Army Research Office MURI DAAD 19-02-01-0383.

[1]A. N. Prior - the father of modern temporal logic - actually believed that tense logic should be related as closely as possible to intuitions embodied in everyday communications.

Ultimately, the human-robot interaction will be part of the every day life. Nevertheless, most of the end users, that is the humans, will not have the required mathematical background in formal methods in order to communicate with the robots. In other words, nobody wants to communicate with a robot using logical symbols - hopefully not even the experts in Linear Temporal Logic. Therefore, in this paper we advocate that structured English should act as a mediator between the logical formalism that the robots accept as input and the natural language that the humans are accustomed to.

From a more practical point of view, structured English helps even the robot savvy to understand better and faster the capabilities of the robot without having an intimate knowledge of the system. This is the case since structured English can be tailored to the capabilities of the robotic system, which eventually restricts the possible sentences in the language. Moreover, since different notations are used for the same temporal operators, a structured English framework targeted for robotic applications can offer a uniform representation of temporal logic formulas. Finally, usage of a controlled language minimizes the problems that are introduced in the system due to ambiguities inherent in natural language [22]. The last point can be of paramount importance in safety-critical applications.

Related research moves along two distinct directions. First, in the context of human-robot interaction through natural language, there has been research that converts natural language input to some form of logic (but not temporal) and then maps the logic statements to basic control primitives for the robot [15], [18]. The authors in [20] show how human actions and demonstrations are translated to behavioral primitives. Note that these approaches lack the mathematical guarantees that our work provides for the composition of the low level control primitives for the motion planning problem. The other direction of research deals with controlled language. In [11], [13], whose application domain is model checking [3], the language is mapped to some temporal logic formula. In [23] it is used to convey user specific spatial representations. In this work we assume the robot has perfect sensors that give it the information it needs. In practice one would have to deal with uncertainties and unknowns. The work in [19] describes a system in which language as well as sensing can be used to get a more reliable description of the world.

## II. PROBLEM FORMULATION

Our goal is to devise a human-robot interface where the humans will be able to instruct the robots in a controlled language environment. The end result of our procedure

should be a set of low level controllers for mobile robots that generate continuous behaviors satisfying the user specifications. Such specifications can depend on the state of the environment as sensed by the robot. Furthermore, they can address both robot motion, i.e. the continuous trajectories, and robot actions, such as making a sound or flashing a light. To achieve this, we need to specify the robot's *workspace and dynamics*, assumptions on *admissible environments*, and the desired *user specification*.

*Robot workspace and dynamics:* We assume that a mobile robot (or possibly several mobile robots) is operating in a polygonal workspace $P$. We partition $P$ using a finite number of convex polygonal regions $P_1, \ldots, P_n$, where $P = \cup_{i=1}^{n} P_i$ and $P_i \cap P_j = \emptyset$ if $i \neq j$. We discretize the position of the robot by creating boolean propositions $Reg = \{r_1, r_2, \ldots, r_n\}$. Here, $r_i$ is true if and only if the robot is located in $P_i$. Since $\{P_i\}$ is a partition of $P$, exactly one $r_i$ is true at any time. We also discretize other actions the robot can perform, such as operating the video camera or transmitter. We denote these propositions as $Act = \{a_1, a_2 \ldots, a_k\}$ which are true if the robot is performing the action and false otherwise. In this paper we assume that such actions can be turned on and off at any time, i.e., there is no minimum or maximum duration for the action. We denote all the propositions that the robot can act upon by $\mathcal{Y} = \{Reg, Act\}$.

*Admissible environments:* The robot interacts with its environment using sensors, which in this paper are assumed to be binary. This is a reasonable assumption to make since decision making in the continuous world always involves some kind of abstraction. We denote the sensor propositions by $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$. An example of such sensor propositions might be `TargetDetected` when the sensor is a vision camera. The user may specify assumptions on the possible behavior of these propositions, thus making implicit assumptions on the behavior of the environment. We guarantee that the robot will behave as desired **only** if the environment behaves as expected, i.e., is admissible, as explained in Section III.

*User Specification:* The desired behavior of the robot is given by the user in structured English. It can include motion, for example "Go to rooms [1, 2, 3] infinitely often". It can include an action that the robot must perform, for example "If you are in room 5, then play music". It can also depend on the environment, for example "If you see Mika, go to room 3 and stay there".

*Problem 1 (From Language to Motion):* Given the robot workspace, initial conditions, and a suitable specification in structured English, construct (if possible) a controller so that the robot's resulting trajectories satisfy the user specification in any admissible environment.

## III. Approach

In this section we give an overview of our approach to creating the desired controller for the robot. Figure 1 shows the three main steps. First, the user specification, together with the environment assumptions and robot workspace and dynamics, are translated into a temporal logic formula $\varphi$.

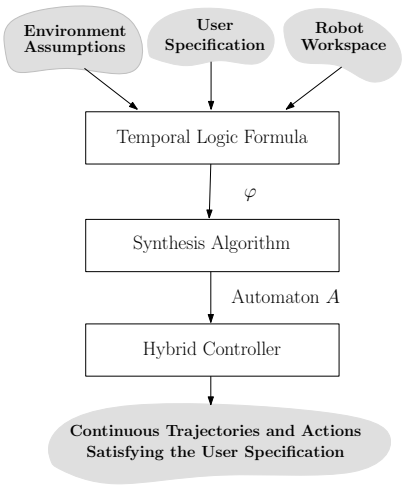



Fig. 1: Overview of the approach

Next, an automaton $A$ that implements $\varphi$ is synthesized. Finally, a hybrid controller based on the the automaton $A$ is created.

The first step, the translation, is the main focus of this paper. In Section IV, we give a detailed description of the logic that we use and in Section VI we show how some behaviors can be automatically translated. For now, let us assume we have constructed the temporal logic formula $\varphi$ and that its atomic propositions are the sensor propositions $\mathcal{X}$ and the robot's propositions $\mathcal{Y}$. The other two steps, i.e. the synthesis of the automaton and creation of the controller, are addressed in [14]. Here, we give a high level description of the process through an illustrative example.

*Hide and Seek:* Our robot is moving in the workspace depicted in Figure 3. It can detect people (through a camera) and it can "beep" (using it's speaker). We want the robot to play "Hide and Seek" with Mika, so we want the robot to search for Mika in rooms 1, 2 and 3. If it sees her, we want it to stay where she is and start beeping. If she disappears, we want the robot to stop beeping and look for her again. We do not assume Mika is willing to play as well. Therefore, if she is not around, we expect the robot to keep looking until we shut it off.

This specification is encoded in a logic formula $\varphi$ that includes the sensor proposition $\mathcal{X} = \{\text{Mika}\}$ and the robot's propositions $\mathcal{Y} = \{r_1, \ldots, r_4, \text{Beep}\}$. The synthesis algorithm outputs an automaton $A$ that implements the desired behavior, *if this behavior can be achieved*. The automaton can be non-deterministic, and is not necessarily unique, i.e. there could be a different automaton that satisfies $\varphi$ as well. The automaton for the *Hide and Seek* example is shown in Figure 2. The circles represent the automaton states and the propositions that are written inside each circle are the robot propositions that are true in that state. The edges are labelled with the sensor propositions that enable that transition, that is a transition labelled with "Mika" can be taken only if Mika is seen. A run of this automaton can start, for example, at the top most state. In this state the robot proposition $r_1$ is true indicating that the robot is in room 1. From there, if the sensor proposition $Mika$ is true a transition is taken to the

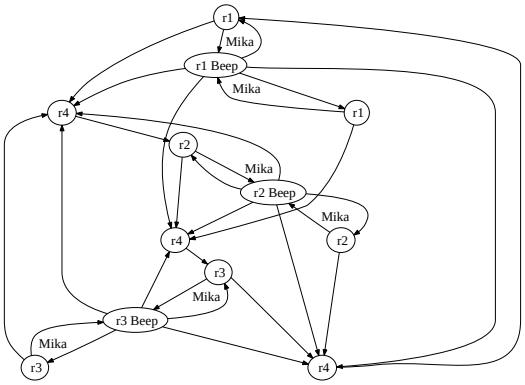
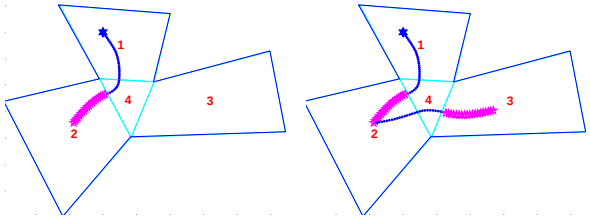



Fig. 2: Automaton for the Hide and Seek example




(a) The robot found Mika in 2 (b) Mika disappeared from 2 and the robot found her again in 3

Fig. 3: Simulation for the Hide and Seek example

state that has both $r_1$ and $Beep$ true meaning that the robot is in room 1 and is beeping, otherwise, a transition is made to the state in which $r_4$ is true indicating the robot is now in room 4 and so on.

The hybrid controller used to drive the robot and control its actions continuously executes the discrete automaton. When the automaton transitions from a state in which $r_i$ is true to a state in which $r_j$ is true, the hybrid controller envokes a simple continuous controller that is gueranteed to drive the robot from $P_i$ to $P_j$ without going through any other cell [1], [6], [17]. Based on the current automaton state, the hybrid controller also activates actions whose propositions are true in that state and deactivates all other robot actions.

Returning to our example, Figure 3 shows a sample simulation. Here Mika is first found in room 2, therefore the robot is beeping (indicated by the lighter colored stars) and staying in that room (Figure 3.a). Then, Mika disappears so the robot stops beeping (indicated by the dark dots) and looks for her again. It finds her in room 3 where it resumes the beeping (Figure 3.b).

## IV. Temporal Logic

We use a fragment of Linear Temporal Logic (LTL) [3] to formally describe the assumptions on the environment, the dynamics of the robot and the desired behavior of the robot, as specified by the user. We first give the syntax and semantics of the full LTL. Then, following [21], we describe the specific structure of the LTL formulas that will be used in this paper.

### A. LTL Syntax and Semantics

*Syntax:* Let $AP$ be a set of atomic propositions. In our setting $AP = \mathcal{X} \cup \mathcal{Y}$, including both sensor and robot propositions. LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following grammar

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

where $\bigcirc$ is the *next* time operator and $\Diamond$ is the *eventually* operator. As usual, the boolean constants $True$ and $False$ are defined as $True = \varphi \vee \neg\varphi$ and $False = \neg True$ respectively. Given negation ($\neg$) and disjunction ($\vee$), we can define conjunction ($\wedge$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). Furthermore, we can also derive the *always* operator as $\Box\varphi = \neg\Diamond\neg\varphi$.

*Semantics:* The semantics of an LTL formula $\varphi$ is defined on an infinite sequence $\sigma$ of truth assignments to the atomic propositions $\pi \in AP$. For a formal definition of the semantics we refer the reader to [3]. Informally, the formula $\bigcirc\varphi$ expresses that $\varphi$ is true in the next "step" (the next position in the sequence). The sequence $\sigma$ satisfies formula $\Box\varphi$ if $\varphi$ is true in every position of the sequence, and satisfies the formula $\Diamond\varphi$ if $\varphi$ is true at some position of the sequence. Sequence $\sigma$ satisfies the formula $\Box\Diamond\varphi$ if $\varphi$ is true infinitely often.

### B. Special class of LTL formulas

Following [21], we consider a special class of temporal logic formulas. These LTL formulas are of the form $\varphi = \varphi_e \Rightarrow \varphi_s$. The formula $\varphi_e$ acts as an assumption about the sensor propositions and, thus, as an assumption about the environment, and $\varphi_s$ represents the desired robot behavior. The formula $\varphi$ is true if $\varphi_s$ is true, i.e., the desired robot behavior is satisfied, **or** $\varphi_e$ is false, i.e., the environment did not behave as expected. This means that when the environment does not satisfy $\varphi_e$ and is thus not admissible, there is no guarantee about the behavior of the robot. Both $\varphi_e$ and $\varphi_s$ have the following structure

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \;\; ; \;\; \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$$

$\varphi_i^e$ and $\varphi_i^s$ describe the initial condition of the environment and the robot. $\varphi_t^e$ represents the assumptions on the environement by constraining the next possible sensor values based on the current sensor and robot values. $\varphi_t^s$ constrains the moves the robot can make and $\varphi_g^e$ and $\varphi_g^s$ represent the assumed goals of the environment and the desired goals of the robot, respectively. For a detailed description of these formulas the reader is referred to [14].

Despite the structural restrictions of this class of LTL formulas, there does not seem to be a significant loss in expressivity as most specifications encountered in practice can be either directly expressed or translated to this format. Furthermore, the structure of the formulas very naturally reflects the structure of most sensor-based robotic tasks.

## V. Environment and Motion Constraints

As mentioned before, we can view the LTL formulas as encoding three components. First, $\varphi^e$ represents the assumptions we make on the behavior of the environment, as sensed

by the robot. Second, $\varphi_i^s$ and $\varphi_t^s$ describe the robot's initial condition and dynamics. Finally, $\varphi_g^s$ represents the desired behavior of the robot. Note that in some cases, the desired behavior is also encoded in $\varphi_t^s$ as discussed in Section VI.

### A. Environment Assumptions

In this paper we allow the user to choose between two types of environments. The first, which is the most general case, is when we have no assumptions on the behavior of the environment, just initial conditions of the sensors. The user input in this case is *"Environment with initial conditions"* **E**"." where E is the set of all sensors that are initially true. In this case

$$\varphi^e_{\text{General}} = \wedge_{x\in \text{E}} x \wedge_{x\notin \text{E}} \neg x \wedge \Box \mathit{True} \wedge \Box\Diamond \mathit{True}$$

The second is the case in which the robot behavior does not depend on it's environment, for example "go to room 4" (no sensing specified). The user input in this case is *"Any Environment."*. Here a dummy sensor proposition must be defined for the completeness of this special class of LTL formulas. We arbitrarily choose it to be always false. Therefore, we have

$$\varphi^e_{\text{NoSensors}} = \neg\text{Dummy} \wedge \Box\neg\text{Dummy} \wedge \Box\Diamond \mathit{True}$$

The logic formulation allows much richer environment assumptions. Creating a language interface for them is a topic for future work.

### B. Motion Constraints

The position of the robot is represented by the propositions $r_i \in \mathcal{Y}$. The robot can only move, at each discrete step, from one cell to an adjacent cell and it can not be in two cells at the same time (mutual exclusion). We can automatically translate these constraints from a description of the workspace into a logic formula. A transition is encoded as

$$\varphi^s_{t_{\text{Transition(i)}}} = \Box(r_i \Rightarrow (\bigcirc r_i \vee_{r\in N} \bigcirc r))$$

where $N$ is the set of all the regions that are adjacent to $r_i$. All transitions are encoded as $\varphi^s_{t_{\text{Transitions}}} = \wedge_{i=1,\ldots,n} \varphi^s_{t_{\text{Transition(i)}}}$. The mutual exclusion is encoded as

$$\varphi^s_{t_{\text{MutualExclusion}}} = \Box(\vee_{1\leq i\leq n} (r_i \wedge_{1\leq j\leq n, i\neq j} \neg r_j))$$

Constraints on the other actions of the robots, if such exist, should be encoded into $\varphi_t^s$ as well. In this paper we assume there are no such constraints.

## VI. Desired Behavior

Our goal in this section is to design a controlled language for the motion and task planning problems for a mobile robot. Similar to [10], [13], we first give a simple grammar (Table I) that produces the sentences in our controlled language and then we give the semantics of some of the sentences in the language with respect to the LTL formulas. We distinguish between two forms of behaviors, *Safety* and *Liveness*. Safety includes all behaviors that the robot must *always* satisfy, such as "Always avoid corridor 2" or "If Mika is found, then stay there". These behaviors are encoded in $\varphi_t^s$ and are of the form $\Box(formula)$. The other behavior, liveness, includes things the robot should *always eventually* satisfy, such as "Go to room 4 infinitely often" or "Go to room 1 infinitely often unless Mika is seen". These behaviors are encoded in $\varphi_g^s$ and are of the form $\Box\Diamond(formula)$.

Some of the rules of the grammar for our controlled language $\mathcal{L}$ appear in Table I. Note that $\mathcal{L}$ is actually an infinite language. The literal terminals are marked using quotation marks "...", the non-literal terminals are denoted by **bold face** (capital letters denote lists of symbols while small letters just one symbol) and non-terminals by *italics*. In some cases, we allow for synonyms in the literal terminals. For example, "go to" can be replaced by "visit" or "reach", while "detected" by "found" or "seen". The terminal **R** ranges over subsets of $Reg$, i.e., over sets of regions of interest. For example **R** can be replaced by {room 1, corridor 2}. **C** ranges over sets of active actions at the initial state. The terminal **s** ranges over the predicates for the sensors, for example "Mika", "fire", "person" and so on, while the terminals $\mathbf{a}_1$, $\mathbf{a}_2$, ... range over predicates for the actions, for example "beep", "picture", "medic", "fireman" and so on. A point that we should make is that the grammar is designed so as the user can write specifications for only one robot. Any inter-robot interaction comes into play through the sensor propisitions. For example we can add a sensor proposition "Robot2in4", which is true whenever the other robot is in room 4, and then refer to that proposition: "If Robot2in4, then go to room 1".

We now show how several simple commands are translated automatically to an LTL formula $\varphi$.

*Initial Conditions:* The initial condition of the robot is given by the user by specifying the initial region that the robot is in and all other output propositions that are initially True. Let $R_r = Reg - \{r\}$, then the sentence "You start in $r$ with initial conditions $C$" is translated to

$$\varphi_i^s = r \wedge_{\bar{r}\in R_r} \neg\bar{r} \wedge_{a\in C} a \wedge_{a\in Act\setminus C} \neg a$$

*Motion Rules:* The requirement "go to $r$ infinitely often" is mapped to the temporal formula:

$$\varphi^s_{g_{\text{GoTo(r)}}} = \Box\Diamond r$$

This formula makes sure the robot visits room $r$ infinitely often. We can request the robot to visit multiple rooms, such as "go to $R'$ infinitely often" for $R' \subseteq Reg$, by taking conjunctions of "go to" specifications. Note that such a conjunction does not specify in which order the rooms must be visited. It only requests that all rooms be visited infinitely often.

The "go to" specification does not make the robot stay in room $r$, once it arrived there. If we want to specify "go to room $r$ and always stay there"[2], we must add a safety behavior that requires the robot to stay in room $r$ once it arrives there. The specification is translated to

$$\varphi^s_{tg_{\text{GoStay(r)}}} = \Box\Diamond r \wedge \Box(r \Rightarrow \bigcirc r)$$

[2]Note that the simple grammar in Table I allows for "go to $r$ infinitely often and go to $q$ and always stay there". This is an infeasible specification, and the synthesis algorithm will inform the user that it is unrealizable.

| | | |
|---|---|---|
| *Start* | ::= | "You start in" **r** "with initial conditions" **C** "." (*Conditional* \| *Motion* "." \| *Motion* "." *Conditional*) |
| *Conditional* | ::= | *Conditional Conditional* \| "If" *Condition* ", then" (*Motion*$^+$ \| *Action*) "." \| |
| | | \| (*Motion*$^+$ \| *Action*) "unless" *Condition* "." \| (*Motion*$^+$ \| *Action*) "iff" *Condition* |
| *Condition* | ::= | *Condition* "and" *Condition* \| *Condition* "or" *Condition* \| "you are in" **R** \| |
| | | \| "you are not in" **R** \| "You detect" **s** \| **s** "is detected" \| ... |
| *Action* | ::= | *Action* "and" *Action* \| *Action*$^+$ |
| *Action*$^+$ | ::= | *Action*$^-$ \| "do not" *Action*$^-$ |
| *Action*$^-$ | ::= | $\mathbf{a}_1$ \| "take" $\mathbf{a}_2$ \| "call" $\mathbf{a}_3$ \| ... |
| *Motion* | ::= | *Motion* "and" *Motion* \| *Motion*$^-$ \| "go to" **r** "and always stay there" |
| *Motion*$^+$ | ::= | *Motion*$^-$ \| "stay there" |
| *Motion*$^-$ | ::= | "go to" **R** "infinitely often" \| "always avoid" **R** \| ... |

TABLE I: The basic grammar rules for the motion planning problem.

This formula states that if the robot is in room $r$, in the next step it must be in room $r$ as well. We define both *Motion* and *Motion*$^+$ to allow sentences of the form "If you sense Mika, then stay there" while prohibiting combinations such as "always avoid $r$ and stay there".

Another motion primitive is avoidance. Since avoidance is a safety behavior, it is encoded in $\varphi^s_t$. The specification "always avoid r" is translated into

$$\varphi^s_{t_{\text{Avoid(r)}}} = \square(\neg \bigcirc r)$$

meaning, the robot will not be in room $r$ in the next step. Again, as before, we can tell the robot to avoid several rooms taking a conjunction of $\varphi^s_{t_{\text{Avoid(r)}}}$

*Conditional Rules:* We can translate "if ... then ..." or "... unless ..." commands using temporal logic by connecting the condition and the requirement with the appropriate logical connective. As an example for a condition, the sentence "you are in $R'$", where $R' \subseteq Reg$, translates to the boolean formula

$$\varphi_{in(R')} = \vee_{r \in R'} r$$

The semantics of the conditional rules depend on the rules used in the consequence. For example, "If condition, then go to $r$" converts to

$$\varphi^s_{g_{\text{IfGoTo(Condition,r)}}} = \square\diamond(\mathit{Condition} \Rightarrow r)$$

While "If condition then avoid $r$" translates to

$$\varphi^s_{t_{\text{IfAvoid}(\mathit{Condition},\text{r})}} = \square(\mathit{Condition} \Rightarrow \neg \bigcirc r)$$

For lack of space we will not discuss further how such conditionals are translated to LTL.

Now we turn to the composition of conditionals with action primitives. Turning on or off other outputs of the robot will typically be a safety behavior of the form "If *on(off)-condition*, then (do not) *action*".

$$\begin{aligned} \varphi^s_{t_{\text{Do(a)}}} &= \square(\ \mathit{OnCondition}\ \Rightarrow \bigcirc a) \\ \varphi^s_{t_{\text{DoNot(a)}}} &= \square(\ \mathit{OffCondition}\ \Rightarrow \neg \bigcirc a) \end{aligned}$$

The conditional "... iff ... " is short for *if and only if* and is created by taking the conjunction of "If" *Condition* ", then" (*Motion*$^+$ | *Action*) "." and "If" NOT *Condition* ", then" NOT (*Motion*$^+$ | *Action*) "."

One final note is that the different sentences in the *Start* rule are converted to a temporal formula by taking conjunctions of the respective temporal subformulas. We give several examples in the next section.

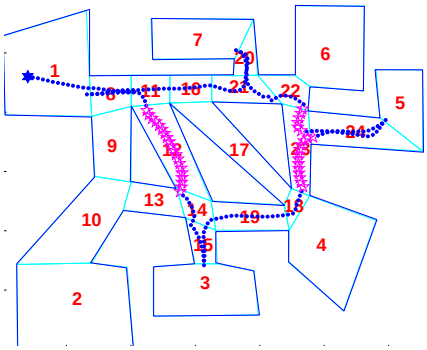



Fig. 4: Simulation for the *Visit and Beep* example

## VII. Examples

In the following, we assume that the workspace of the robot contains 24 rooms (Figures 4, 5). Given this workspace we automatically generate $\varphi^s_{t_{\text{Transitions}}}$ and $\varphi^s_{t_{\text{MutualExclusion}}}$ relating to the motion constraints

**No Sensors:** Here we assume the robot has no sensor inputs, therefore we will automatically generate the `dummy` proposition and $\varphi^e = \varphi^e_{\text{NoSensors}}$

*Visit and Beep:* In this example the robot can move and beep, therefore $\mathcal{Y} = \{r_1, \ldots, r_{24}, \mathit{Beep}\}$. The user specification is: "Any Environment. You start in $r_1$ with initial conditions $\emptyset$. Go to $\{r_1, r_3, r_5, r_7\}$ infinitely often. Beep iff you are in $\{r_9, r_{12}, r_{17}, r_{23}\}$."

The behavior of the above example is first automatically translated into the formula $\varphi$:

$$\begin{aligned} \varphi^e &= \neg\text{Dummy} \wedge \square\neg\text{Dummy} \wedge \square\diamond \mathit{True} \\ \varphi^s &= \begin{cases} r_1 \wedge_{i=2,\ldots,24} \neg r_i \wedge \neg\text{Beep} \\ \wedge\varphi^s_{t_{\text{Transitions}}} \wedge \varphi^s_{t_{\text{MutualExclusion}}} \\ \wedge\square\diamond(r_1) \wedge \square\diamond(r_3) \wedge \square\diamond(r_5) \wedge \square\diamond(r_7) \\ \wedge\square((r_9 \vee r_{12} \vee r_{17} \vee r_{23}) \Rightarrow \bigcirc\text{Beep}) \\ \wedge\square(\neg(r_9 \vee r_{12} \vee r_{17} \vee r_{23}) \Rightarrow \neg \bigcirc \text{Beep}) \end{cases} \end{aligned}$$

Then an automaton is synthesized and a hybrid controller is constructed. Sample simulations are shown in Figure 4. As before, beeping is indicated by lighter colored stars.

**Sensors:** Let us assume that the robot has two sensors, a camera that can detect an injured person and another sensor that can detect a gas leak, therefore $\mathcal{X} = \{\text{Person}, \text{Gas}\}$.

*Search and Rescue:* Here, other than moving, the robot can communicate to the base station a request for either a medic or a fireman. We assume that the base station can track the robot therefore it does not need to transmit it's location. We

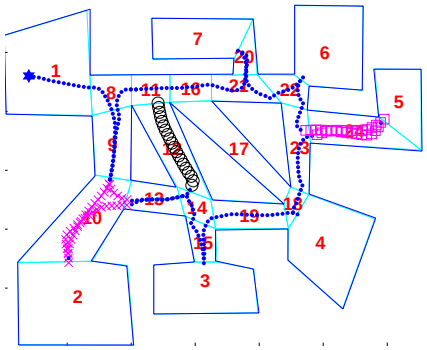



Fig. 5: Simulation for the *Search and Rescue* example

define $\mathcal{Y} = \{r_1, \ldots, r_{24}, \text{Medic}, \text{Fireman}\}$. The user specification is "Environment with initial conditions $\emptyset$. You start in $r_1$ with initial conditions $\emptyset$. Go to $\{r_1, \ldots, r_{24}\}$ infinitely often. Call Medic iff Person is found. Call Fireman iff Gas is detected."

A sample simulation is shown in Figure 5. Here, a person was detected in region 10 resulting in a call for a Medic (light cross). A gas leak was detected in region 24 resulting in a call for a Fireman (light squares). In region 12, both a person and a gas leak were detected resulting in a call for both a Medic and a Fireman (dark circles)

## VIII. Conclusions - Future work

In this paper we have described a method for automatically translating robot behaviors from a user specified description in structured English to actual robot controllers and trajectories. Furthermore, this framework allows the user to specify reactive behaviors that depend on the information the robot gathers from its environment at run time. We have shown how several complex robot behaviors can be expressed using structured English and how these phrases can be translated into temporal logic. The extension of the results in this paper to deal with complex dynamics [7] as well as non-holonomic vehicles [5] follows naturally.

As mentioned in this paper, we have not yet captured the full expressive power of the special class of LTL formulas. This logic allows the user to specify sequences of behaviors, different environment assumptions and other robot constraints. This is a topic of future work.

We also intend to construct a corpus of what people would typically ask a robot to do and use it to explore if and how natural language might be translated into the logic formulation.

## IX. Acknowledgments

We would like to thank David Conner for allowing us to use his code for the potential field controllers and Nir Piterman, Amir Pnueli and Yaniv Sa'ar for allowing us to use their code for the synthesis algorithm.

## References

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

[2] H. Choset, K. M. Lynch, L. Kavraki, W. Burgard, S. A. Hutchinson, G. Kantor, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, USA, 2005.

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

[4] D. C. Conner, H. Choset, and A. Rizzi. Towards provable navigation and control of nonholonomically constrained convex-bodied systems. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA '06)*, May 2006.

[5] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas. Valet parking without a valet. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, San Diego, CA, October 2007.

[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] G. E. Fainekos, A. Girard, and G. J. Pappas. Hierarchical synthesis of hybrid controllers from temporal logic specifications. In *Hybrid Systems: Computation and Control*, number 4416 in LNCS, page 203216. Springer, 2007.

[8] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, pages 4885–4890, Seville, Spain, 2005.

[9] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 2020–2025, Barcelona, Spain, 2005.

[10] S. Flake, W. Müller, and J. Ruf. Structured english for model checking specification. In *GI–Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen in Frankfurt*, Berlin, 2000. VDE Verlag.

[11] A. Holt and E. Klein. A semantically-derived subset of english for hardware verification. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 451–456, Morristown, NJ, USA, 1999. Association for Computational Linguistics.

[12] M. Kloetzer and C. Belta. Hierarchical abstractions for robotic swarms. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 952 – 957, 2006.

[13] S. Konrad and B. H. C. Cheng. Facilitating the construction of specification pattern-based properties. In *Proceedings of the IEEE International Requirements Engineering Conference*, Paris, France, August 2005.

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

[15] S. Lauria, T. Kyriacou, G. Bugmann, J. Bos, and E. Klein. Converting natural language route instructions into robot-executable procedures. In *Proceedings of the 2002 IEEE International Workshop on Robot and Human Interactive Communication*, pages 223–228, Berlin, 2002.

[16] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[17] S. Lindemann and S. LaValle. Computing smooth feedback plans over cylindrical algebraic decompositions. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[18] A. J. Martignoni III and W. D. Smart. Programming robots using high-level task descriptions. In M. Rosenstein and M. Ghavamzadeh, editors, *Proceedings of the AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, pages 49–54, June 2004.

[19] N. Mavridis and D. Roy. Grounded situation models for robots: Where words and percepts meet. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, Beijing, China, October 2006.

[20] M. Nicolescu and M. J. Mataric. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics, Part B,special issue on Socially Intelligent Agents - The Human in the Loop*, 31(5):419–430, 2001.

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

[22] S. Pulman. Controlled language for knowledge representation. In *Proceedings of the 1st International Workshop on Controlled Language Applications*, 1996.

[23] E. A. Topp, H. Hüttenrauch, H. I. Christensen, and K. S. Eklundh. Bringing together human and robotics environmental representations / a pilot study. In *Proc. IEEE/RSJ Intl Conf on Intell. Robots and Systems (IROS-06)*, Beijing, CH, October 2006.