Advanced Robotics

ISSN: 0169-1864 (Print) 1568-5535 (Online) Journal homepage: https://www.tandfonline.com/loi/tadr20

Translating Structured English to Robot Controllers

Hadas Kress-Gazit , Georgios E. Fainekos & George J. Pappas

To cite this article: Hadas Kress-Gazit , Georgios E. Fainekos & George J. Pappas (2008) Translating Structured English to Robot Controllers, Advanced Robotics, 22:12, 1343-1359, DOI: 10.1163/156855308X344864

To link to this article: https://doi.org/10.1163/156855308X344864

Published online: 02 Apr 2012.

VSP

Advanced Robotics 22 (2008) 1343–1359

ADVANCED ROBOTICS
www.brill.nl/ar

Full paper

Translating Structured English to Robot Controllers

Hadas Kress-Gazit *, **Georgios E. Fainekos and George J. Pappas**

GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA

Received 1 February 2008; accepted 8 May 2008

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.

Keywords

Motion planning, task planning, temporal logic, structured English

1. 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 Refs [1, 2]). Recent advances [3–6] 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 [1, 2]. One such framework is based on automata theory where the specification language is the so-called Linear Temporal Logic (LTL) [7]. 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 [10]. 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

* To whom correspondence should be addressed. E-mail: hadaskg@grasp.upenn.edu

 DOI:10.1163/156855308X344864

a reactive way. In our recent work [11], we have shifted to a framework that solves the planning problem for a fragment of LTL [12], 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 (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). 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 LTL 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.

Ultimately, the human–robot interaction will be part of every day life. Nevertheless, most of the end-users, i.e., 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 experts in LTL. 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 to which humans are accustomed.

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 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 [13]. 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 [14, 15]. The authors in Ref. [16] 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 Refs [17, 18], whose application domain is model checking [7], the language is mapped to some temporal logic formula. In Ref. [19] 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 Ref. [20] describes a system in

which language as well as sensing can be used to get a more reliable description of the world.

2. 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 $P = \bigcup_{i=1}^{n} P_i$ 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 3.

User specification. The desired behavior of the robot is given by the user in structured English. It can include motion, e.g., 'go to room_1 and go to room_2 and go to room_3 or room_4'. It can include an action that the robot must perform, e.g., 'if you are in room_5 then play music'. It can also depend on the environment, e.g., 'if you are sensing Mika then 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)*

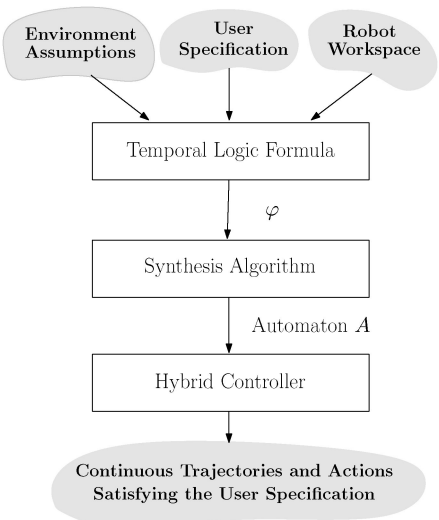


Figure 1. Overview of the approach.

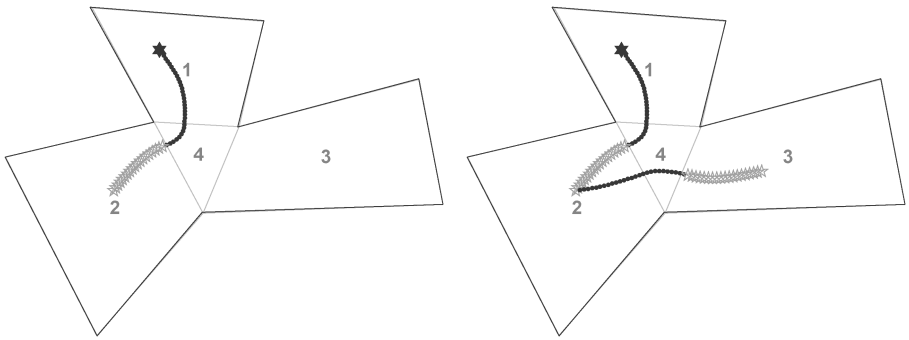
a controller so that the robot's resulting trajectories satisfy the user specification in any admissible environment.

3. 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 φ. Next, an automaton A that implements φ 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 4, we give a detailed description of the logic that is used and in Section 5 we show how some behaviors can be automatically translated. For now, let us assume we have constructed the temporal logic formula φ, 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 Ref. [11]. 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 Fig. 2. It can detect people (through a camera) and it can 'beep' (using its 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.



(a) (b)

Figure 2. Simulation 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.

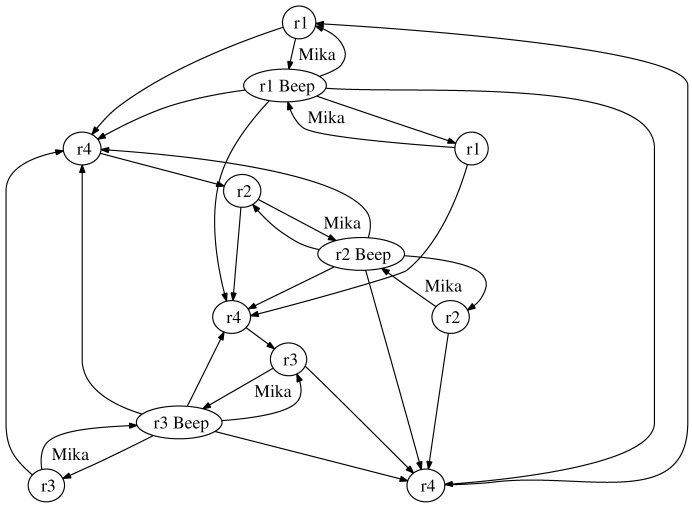


Figure 3. Automaton for the Hide and Seek example.

This specification is encoded in a logic formula φ that includes the sensor proposition $\mathcal{X} = \{Mika\}$ and the robot's propositions $\mathcal{Y} = \{r_1, \ldots, r_4, Beep\}$. 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 φ as well. The automaton for the Hide and Seek example is shown in Fig. 3. 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 labeled with the sensor propositions that enable that transition, i.e., a transition labeled 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 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 guaranteed to drive the robot from P_i to P_j without going through any other cell [3, 5, 6]. 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, Fig. 2 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 (Fig. 2a). 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 (Fig. 2b).

4. Temporal Logic

We use a fragment of LTL [7] 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 Ref. [12], we describe the specific structure of the LTL formulas that will be used in this paper.

4.1. LTL Syntax and Semantics

4.1.1. 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 $\Box\varphi = \neg\diamond\neg\varphi$.

4.1.2. Semantics

The semantics of an LTL formula φ is defined on an infinite sequence σ of truth assignments to the atomic propositions $\pi \in AP$. For a formal definition of the semantics we refer the reader to Ref. [7]. Informally, the formula $\bigcirc\varphi$ expresses that φ is true in the next 'step' (the next position in the sequence). The sequence σ satisfies formula $\Box\varphi$ if φ is true in every position of the sequence and satisfies the

formula $\diamond\varphi$ if φ is true at some position of the sequence. Sequence σ satisfies the formula $\square\diamond\varphi$ if φ is true infinitely often.

4.2. Special Class of LTL Formulas

Following Ref. [12], we consider a special class of temporal logic formulas. These LTL formulas are of the form $\varphi = \varphi_e \Rightarrow \varphi_s$. The formula φ_e acts as an assumption about the sensor propositions and, thus, as an assumption about the environment, and φ_s represents the desired robot behavior. The formula φ is true if φ_s is true, i.e., the desired robot behavior is satisfied, or φ_e is false, i.e., the environment did not behave as expected. This means that when the environment does not satisfy φ_e and is, thus, not admissible, there is no guarantee about the behavior of the robot. Both φ_e and φ_s have the following structure:

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

where φ_i^e and φ_i^s describe the initial condition of the environment and the robot, φ_t^e represents the assumptions on the environement by constraining the next possible sensor values based on the current sensor and robot values, φ_t^s constrains the moves the robot can make, and φ_g^e and φ_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 Ref. [11].

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.

5. Structured English

Our goal in this section is to design a controlled language for the motion and task planning problems for a mobile robot. Similar to Ref. [18, 21], we first give a simple grammar (Table 1) 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 described in Section 4.

5.1. Grammar

First we define a set of Boolean formulas:

$$\phi ::= x \in \mathcal{X} \mid y \in \mathcal{Y} \mid \text{not } \phi \mid \phi \text{ or } \phi \mid \phi \text{ and } \phi \mid \phi \text{ implies } \phi \mid \phi \text{ iff } \phi$$

$$\phi_{env} ::= x \in \mathcal{X} \mid \text{not } \phi_{env} \mid \phi_{env} \text{ or } \phi_{env} \mid \phi_{env} \text{ and } \phi_{env} \mid \phi_{env} \text{ implies } \phi_{env} \mid \phi_{env} \text{ iff } \phi_{env}$$

$$\phi_{robot} ::= y \in \mathcal{Y} \mid \text{not } \phi_{robot} \mid \phi_{robot} \text{ or } \phi_{robot} \mid \phi_{robot} \text{ and } \phi_{robot} \mid \phi_{robot} \text{ implies } \phi_{robot} \mid \phi_{robot} \text{ iff } \phi_{robot}$$

Table 1.
The basic grammar rules for the motion planning problem

EnvInit ::= 'Environment starts with (ϕ_{env} | false | true)'
RobotInit ::= 'Robot starts [in ϕ_{region}] [with ϕ_{action} | with false | with true]'
EnvSafety ::= 'Always ϕ_{env}'
RobotSafety ::= '(Always | Always do | Do) ϕ_{robot}'
EnvLiveness ::= 'Infinitely often ϕ_{env}'
RobotLiveness ::= '(Go to | Visit | Infinitely often do) ϕ_{robot}'
RobotGoStay ::= 'Go to ϕ_{region} and stay [there]'
Conditional ::= 'If *Condition* then *Requirement* ' | '*Requirement* unless *Condition*' | '*Requirement* if and only if *Condition*'
Condition ::= '*Condition* and *Condition*' | '*Condition* or *Condition*' | 'you (were | are) [not] in ϕ_{region}' | 'you (sensed | did not sense | are [not] sensing) ϕ_{env}' | 'you (activated | did not activate | are [not] activating) ϕ_{action}' |
Requirement ::= *EnvSafety* | *RobotSafety* | *EnvLiveness* | *RobotLiveness* | 'stay [there]'

$$\phi_{\text{region}} ::= r \in \mathit{Reg} \mid \text{not } \phi_{\text{region}} \mid \phi_{\text{region}} \text{ or } \phi_{\text{region}} \mid \phi_{\text{region}} \text{ and } \phi_{\text{region}} \mid \phi_{\text{region}} \text{ implies } \phi_{\text{region}} \mid \phi_{\text{region}} \text{ iff } \phi_{\text{region}}$$

$$\phi_{\text{action}} ::= a \in \mathit{Act} \mid \text{not } \phi_{\text{action}} \mid \phi_{\text{action}} \text{ or } \phi_{\text{action}} \mid \phi_{\text{action}} \text{ and } \phi_{\text{action}} \mid \phi_{\text{action}} \text{ implies } \phi_{\text{action}} \mid \phi_{\text{action}} \text{ iff } \phi_{\text{action}}.$$

Intuitively, ϕ captures a logical connection between propositions belonging to both the environment and the robot, while ϕ_{env} restricts to propositions relating to the environment, ϕ_{robot} restricts to propositions relating to the robot, ϕ_{region} to to the robot's region propositions and ϕ_{action} to the robot's action propositions.

The grammar in Table 1 contains different types of sentences. In each of these, exactly one of the terms written inside of parentheses is required while terms written inside square brackets are optional. Past and present tenses in *Condition* are treated differently only when combined with *EnvSafety* or *RobotSafety* or *Stay* as explained in Section 5.4. In all other cases, they are treated the same. The user specification may include any combination of sentences and there is no minimal set of instructions that must be written. If some sentences, such as initial conditions, are omitted, their corresponding formulas are replaced by default values as defined in the next sections.

We distinguish between two forms of behaviors, *Safety* and *Liveness*. Safety includes all behaviors that the environment or the robot must always satisfy, such as not sensing Mika in a certain region or avoiding regions. These behaviors are encoded in $\varphi_{\text{t}}^{\text{e}}$ and $\varphi_{\text{t}}^{\text{s}}$ and are of the form $\Box(\mathit{formula})$. The other behavior, liveness,

includes things the environment or the robot should always eventually satisfy, such as sensing Mika (if we assume she is somewhere in the environment and not constantly avoiding the robot) or reaching rooms. These behaviors are encoded in φ_g^e and φ_g^s, and are of the form $\square\diamond(\textit{formula})$.

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 propositions. 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 you are sensing Robot2in4 then go to room1'.

The different sentences are converted to a single temporal logic formula by taking conjunctions of the respective temporal subformulas. We give several examples in Section 6.

5.2. *Initial Conditions:* EnvInit, RobotInit

The formula capturing the assumed initial condition of the environment φ_i^e is a Boolean combination of propositions in $\mathcal{X}$ while the initial condition for the robot φ_i^s is a Boolean combination of propositions in $\mathcal{Y}$. Therefore, the different options in *EnvInit* are translated to $\varphi_i^e = \phi_{\text{env}}$, $\varphi_i^e = \bigwedge_{x\in\mathcal{X}} \neg x$ and $\varphi_i^e = \bigwedge_{x\in\mathcal{X}} x$, respectively. *RobotInit* is translated in the same way.

If *EnvInit* or *RobotInit* are not specified, then the corresponding formula is set to true, thus allowing the initial values of the propositions to be unknown, meaning the robot can start anywhere and can sense anything upon waking up. Furthermore, if ϕ_{env} or ϕ_{region} or ϕ_{action} do not contain all the relevant propositions, the set of valid initial conditions contain all possible combinations of truth values for the omitted propositions.

5.3. *Liveness:* EnvLiveness, RobotLiveness, RobotGoStay

The subformulas φ_g^e and φ_g^s capture the assumed liveness properties of the environment and the desired goals of the robot. The basic liveness specifications, *EnvLiveness* and *RobotLiveness*, translate to $\varphi_g^e = \square\diamond(\phi_{\text{env}})$ and $\varphi_g^s = \square\diamond(\phi_{\text{robot}})$, respectively. These formulas specify that infinitely often, ϕ_{env} and ϕ_{robot} must hold.

The 'go to' specification does not make the robot stay in room r once it arrives there. If we want to specify 'go to r and stay there', we must add a safety behavior that requires the robot to stay in room r once it arrives there. Note that the simple grammar in Table 1 allows for 'go to r' and 'go to q and stay there'. This is an unfeasible specification since 'go to' implies visiting a region infinitely often and the synthesis algorithm will inform the user that it is unrealizable. The specification is translated to:

$$\varphi^s_{\text{tg}_{GoStay(r)}} = \square\diamond r \wedge \square(r \Rightarrow \bigcirc r).$$

This formula states that if the robot is in room r, in the next step it must be in room r as well.

The general form of a liveness conditional is translated into:

$$\text{If statement: } \varphi_t^{\alpha} = \Box\Diamond(\mathit{Condition} \Rightarrow \mathit{Requirement})$$
$$\text{Unless statement: } \varphi_t^{\alpha} = \Box\Diamond(\mathit{Condition} \vee \mathit{Requirement})$$
$$\text{If and only if statement: } \varphi_t^{\alpha} = \Box\Diamond(\mathit{Condition} \Leftrightarrow \mathit{Requirement}),$$

where $\alpha \in \{e, s\}$, the condition is captured by a combination of ϕ_{env}, ϕ_{region} and ϕ_{action} and the requirement is captured by (ϕ_{env}) or (ϕ_{robot}).

If there are no liveness requirements for the environment or for the robot then the corresponding formula is set to $\Box\Diamond(\text{true})$.

5.4. Safety: EnvSafety, RobotSafety

The subformulas φ_t^e and φ_t^s capture how the sensor and robot propositions are allowed to change from the current state in the synthesized automaton to the next. Due to the way the algorithm works (see [11, 12]), the next sensor values can only depend on the current sensor and robot values and the next sensor values while the next robot values can depend on the current as well as the next sensor and robot values. (The sensing is done before acting, therefore when choosing the next motion and action for the robot, the most recent sensor information can be used.)

The basic safety specifications, *EnvSafety* and *RobotSafety*, translate to $\varphi_t^e = \Box(\bigcirc(\phi_{env}))$ and $\varphi_t^s = \Box(\bigcirc(\phi_{robot}))$, respectively. These formulas specify that always, at the next state, ϕ_{env} and ϕ_{robot} must hold.

The general form of a safety conditional is translated into:

$$\text{If statement: } \varphi_t^{\alpha} = \Box(\mathit{Condition} \Rightarrow \mathit{Requirement})$$
$$\text{Unless statement: } \varphi_t^{\alpha} = \Box(\mathit{Condition} \vee \mathit{Requirement})$$
$$\text{If and only if statement: } \varphi_t^{\alpha} = \Box(\mathit{Condition} \Leftrightarrow \mathit{Requirement}),$$

where $\alpha \in \{e, s\}$. The requirement is captured by $\bigcirc(\phi_{env})$ or $\bigcirc(\phi_{robot})$ or 'Stay [there]' which is translated to the formula $\bigwedge_{r \in Reg}(r \Leftrightarrow \bigcirc r)$ that encodes the requirement that the robot not change the region it is in (if r_i is true that it is also true in the next state).

As mentioned before, for safety conditions, the past tense relates to the value of the propositions in the current state while the present tense relates to the value of the propositions in the next state. The atomic conditions are translated into:

$$\phi \text{ positive past}$$
$$\neg\phi \text{ negative past}$$
$$\bigcirc\phi \text{ positive present}$$
$$\neg \bigcirc \phi \text{ negative present},$$

where ϕ is either ϕ_{env} or ϕ_{region} or ϕ_{action}, depending on the sentence. A general condition can be built up by combining several atomic conditions using 'and' ($\wedge$) and 'or' ($\vee$).

One should keep in mind that the past tense only relates to situations that are true in the current state. Such behaviors may include '`If you were in` r_1 `and you are in` r_2 `then` Beep', meaning that if the robot moved from region 1 to region 2 it should beep. This sentence does not capture a behavior in which if the robot was sometime in the past in region 1 and now it is in region 2 it should beep. If the desired behavior needs to depend on events that took place in the past, an auxiliary proposition should be created to 'remember' that the event took place.

In addition to the structured English sentences, we automatically encode motion constraints on the robot that are induced by the workspace decomposition. These constraints are that the robot can only move, at each discrete step, from one region to an adjacent region and it cannot be in two regions at the same time (mutual exclusion). A transition is encoded as:

$$\varphi^{\mathrm{s}}_{\mathrm{t}_{Transition(i)}} = \Box(r_i \Rightarrow (\bigcirc r_i \vee_{r\in N} \bigcirc r)),$$

where N is the set of all the regions that are adjacent to r_i. All transitions are encoded as:

$$\varphi^{\mathrm{s}}_{\mathrm{t}_{Transitions}} = \wedge_{i=1,\ldots,n} \varphi^{\mathrm{s}}_{\mathrm{t}_{Transition(i)}}.$$

The mutual exclusion is encoded as:

$$\varphi^{\mathrm{s}}_{\mathrm{t}_{MutualExclusion}} = \Box(\vee_{1\leqslant i\leqslant n} (\bigcirc r_i \wedge_{1\leqslant j\leqslant n, i\neq j} \neg \bigcirc r_j)).$$

If there are no safety requirements for the environment then $\varphi^{\mathrm{e}}_{\mathrm{t}} = \Box(\mathrm{true})$, thus allowing the sensor propositions to become true or false at any time. If there are no safety requirements for the robot then $\varphi^{\mathrm{s}}_{\mathrm{t}} = \varphi^{\mathrm{s}}_{\mathrm{t}_{Transitions}} \wedge \varphi^{\mathrm{s}}_{\mathrm{t}_{MutualExclusion}}$, encoding only the workspace decomposition.

6. Examples

In the following, we assume that the workspace of the robot contains 24 rooms (Figs 4 and 5). Given this workspace we automatically generate $\varphi^{\mathrm{s}}_{\mathrm{t}_{Transitions}}$ and $\varphi^{\mathrm{s}}_{\mathrm{t}_{MutualExclusion}}$ relating to the motion constraints.

6.1. No Sensors

The behavior of the robot in these examples does not depend on its sensor inputs. However, for the completeness of the LTL formula, at least one sensor input must be specified. Therefore, we create a dummy sensor input $\mathcal{X} = \{Dummy\}$ which we define to be constant in order to reduce the size of the automaton. We arbitrarily choose it to be false.

6.1.1. Visit and Beep

Here the robot can move and beep; therefore, $\mathcal{Y} = \{r_1, \ldots, r_{24}, Beep\}$. The desired behavior of the robot includes visiting rooms 1, 3, 5 and 7 infinitely often, and beeping whenever it is in corridors 9, 12, 17 and 23, but only there. We assume the robot does not initially beep. The user specification is:

* 'Environment starts with false'
* 'Always not *Dummy*'
* 'Robot starts in r_1 with false'
* 'Activate *Beep* if and only if you are in r_9 or r_{12} or r_{17} or r_{23}'
* 'Go to r_1'
* 'Go to r_3'
* 'Go to r_5'
* 'Go to r_7'.

The behavior of the above example is first automatically translated into the formula φ:

$$\varphi^{\mathrm{e}} = \neg \mathit{Dummy} \wedge \Box\neg \mathit{Dummy} \wedge \Box\Diamond \mathit{True}$$

$$\varphi^{\mathrm{s}} = \begin{cases} r_1 \wedge_{i=2,\ldots,24} \neg r_i \wedge \neg \mathit{Beep} \\ \wedge \varphi^{\mathrm{s}}_{\mathrm{t}_{\mathit{Transitions}}} \wedge \varphi^{\mathrm{s}}_{\mathrm{t}_{\mathit{MutualExclusion}}} \\ \wedge \Box((r_9 \vee r_{12} \vee r_{17} \vee r_{23}) \Leftrightarrow \bigcirc \mathit{Beep}) \\ \wedge \Box\Diamond(r_1) \wedge \Box\Diamond(r_3) \wedge \Box\Diamond(r_5) \wedge \Box\Diamond(r_7). \end{cases}$$

Then an automaton is synthesized and a hybrid controller is constructed. Sample simulations are shown in Fig. 4. As in the Hide and Seek example of Section 3, beeping is indicated by lighter colored stars.

6.1.2. Visit While Avoiding — Impossible Specification
Here, we assume the robot can only move. The user specification is:

* 'Environment starts with false'
* 'Always not *Dummy*'
* 'Robot starts in r_1 with false'
* 'Always not r_8 and not r_{22}'

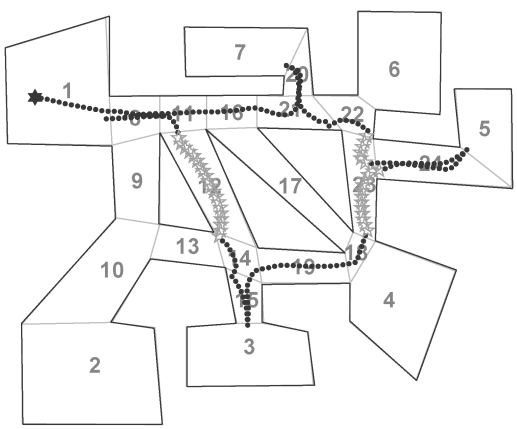


Figure 4. Simulation for the visit and beep example.

∗ 'Go to r_2'
∗ 'Go to r_5'.

Viewing the workspace of the robot we can see that this user specification is impossible to implement since the robot cannot leave room 1 if it avoids 8 and, thus, it cannot reach the other room. In this case, the synthesis algorithm terminates with the message that the specification is not realizable.

6.2. Sensors

6.2.1. Search and Rescue I

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, $\mathcal{X} = \{Person, Gas\}$. 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 its location. We define $\mathcal{Y} = \{r_1, \ldots, r_{24}, Medic, Fireman\}$. The user specification is:

∗ 'Environment starts with false'
∗ 'Robot starts in r_1 with false'
∗ 'Do *Medic* if and only if you are sensing *Person*'
∗ 'Do *Fireman* if and only if you are sensing *Gas*'
∗ 'Go to r_1'
⋮
∗ 'Go to r_{24}'.

A sample simulation is shown in Fig. 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

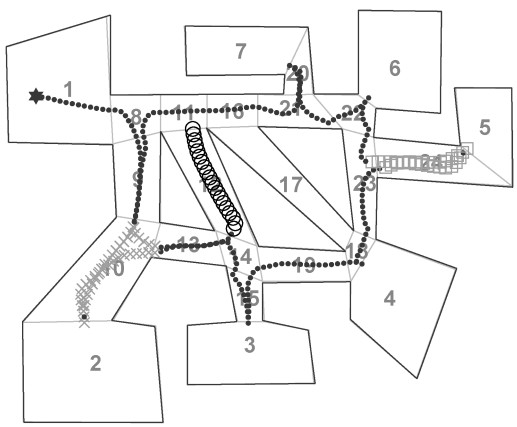


Figure 5. Simulation for the search and rescue I example.

a gas leak were detected resulting in a call for both a *Medic* and a *Fireman* (dark circles).

6.2.2. Search and Rescue II

In this scenario, the robot must monitor rooms 1, 3, 5 and 7 for injured people (we assume that injured people can only appear in these rooms). If it senses an injured person, the robot must pick him or her up and take them to a medic who is in either room 2 or room 4. Once the robot finds the medic, it leaves the injured person there and continues to monitor the rooms. We define $\mathcal{X} = \{Person, Medic\}$ and $\mathcal{Y} = \{r_1, \ldots, r_{24}, carryPerson\}$. The specification is:

* 'Environment starts with false'
* 'If you were not in r_1 or r_3 or r_5 or r_7 then always not *Person*'
* 'If you were not in r_2 or r_4 then always not *Medic*'
* 'Robot starts in r_1 with false'
* 'If you are sensing *Person* then do *carryPerson*'
* 'If you are not sensing *Person* and you did not activate *carryPerson* then do not *carryPerson*'
* 'If you are not sensing *Medic* and you activated *carryPerson* then do *carryPerson*'
* 'If you are sensing *Medic* and you activated *carryPerson* then do not *carryPerson*'
* 'Go to r_1 unless you are activating *carryPerson*'
* 'Go to r_3 unless you are activating *carryPerson*'
* 'Go to r_5 unless you are activating *carryPerson*'
* 'Go to r_7 unless you are activating *carryPerson*'
* 'If you are activating *carryPerson* then go to r_2'
* 'If you are activating *carryPerson* then go to r_4'.

A sample simulation is shown in Fig. 6. In the simulation, the robot begins by searching rooms 1 then 3 then 5 and finally 7 for an injured person (Fig. 6a). It finds one in region 7 and goes looking for a *Medic* (indicated by the light crosses) first in room 2 and then in room 4. It finds the *Medic* in room 4 and starts to head back through region 17 towards region 1 to continue its task of monitoring (Fig. 6b).

7. Conclusions and 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.

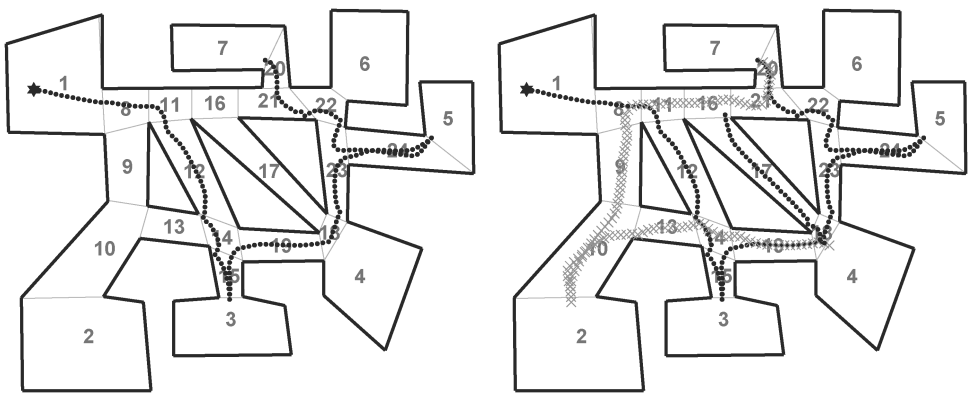


Figure 6. Simulation for the search and rescue II example. (a) The robot search the rooms and found an injured person in room 7. (b) The medic was found in room 4.

This paper focuses on the first step of the approach described in Section 3, the translation of the language to the logic, and does not elaborate on other aspects of this method. Real-world robotics issues such as dealing with complex dynamics, non-holonomic constraints and control robustness were addressed in Refs [22, 23] and are active research directions as is dealing with sensor noise and uncertainty.

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 and add memory propositions among other things. We intend to explore how more complex behaviors can be specified in this framework by both enriching the structured English constructs and exploring how results from computational linguistics can be adapted to this domain.

Another direction we intend to pursue is usability and user interface. Currently, the user is alerted if the a sentence is not in the correct format and cannot be parsed. We intend to examine how tools such as auto completion, error highlighting and templates can facilitate the process of writing the specifications.

Acknowledgements

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. 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, Cambridge, MA (2005).
2. S. M. LaValle, *Planning Algorithms*. Cambridge University Press, Cambridge (2006). Available online at http://planning.cs.uiuc.edu/

3. C. Belta and L. Habets, Constructing decidable hybrid systems with velocity bounds, in: *Proc. IEEE Conf. on Decision and Control*, Bahamas, pp. 467–472 (2004).
4. D. C. Conner, H. Choset and A. Rizzi, Towards provable navigation and control of nonholonomically constrained convex-bodied systems, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Orlando, FL, pp. 57–64 (2006).
5. D. C. Conner, A. A. Rizzi and H. Choset, Composition of local potential functions for global robot control and navigation, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, pp. 3546–3551 (2003).
6. S. Lindemann and S. LaValle, Computing smooth feedback plans over cylindrical algebraic decompositions, in: *Proc. Robotics: Science and Systems*, Cambridge, MA, pp. 207–214 (2006).
7. E. M. Clarke, O. Grumberg and D. A. Peled, *Model Checking*. MIT Press, Cambridge, MA (1999).
8. G. E. Fainekos, H. Kress-Gazit and G. J. Pappas, Hybrid controllers for path planning: a temporal logic approach, in: *Proc. IEEE Conf. on Decision and Control*, Seville, pp. 4885–4890 (2005).
9. G. E. Fainekos, H. Kress-Gazit and G. J. Pappas, Temporal logic motion planning for mobile robots, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Barcelona, pp. 2020–2025 (2005).
10. M. Kloetzer and C. Belta, Hierarchical abstractions for robotic swarms, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Orlando, FL, pp. 952–957 (2006).
11. H. Kress-Gazit, G. E. Fainekos and G. J. Pappas, Where's Waldo? sensor based temporal logic motion planning, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Rome, pp. 3116–3121 (2007).
12. N. Piterman, A. Pnueli, and Y. Sa'ar, Synthesis of reactive(1) designs, in: *Proc. VMCAI*, Charleston, SC, pp. 364–380 (2006).
13. S. Pulman, Controlled language for knowledge representation, in: *Proc. 1st Int. Workshop on Controlled Language Applications*, Leuven, pp. 233–242 (1996).
14. S. Lauria, T. Kyriacou, G. Bugmann, J. Bos and E. Klein, Converting natural language route instructions into robot-executable procedures, in: *Proc. IEEE Int. Workshop on Robot and Human Interactive Communication*, Berlin, pp. 223–228 (2002).
15. A. J. Martignoni, III and W. D. Smart, Programming robots using high-level task descriptions, in: *Proc. AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, San Jose, CA, pp. 49–54 (2004).
16. M. Nicolescu and M. J. Mataric, Learning and interacting in human-robot domains, *IEEE Trans. Syst. Man Cybernet. B* (Special Issue on Socially Intelligent Agents — The Human in the Loop) **31**, 419–430 (2001).
17. A. Holt and E. Klein, A semantically-derived subset of english for hardware verification, in: *Proc. 37th Annu. Meet. of the Association for Computational Linguistics on Computational Linguistics*, Morristown, NJ, pp. 451–456 (1999).
18. S. Konrad and B. H. C. Cheng, Facilitating the construction of specification pattern-based properties, in: *Proc. IEEE Int. Requirements Engineering Conf.*, Paris, pp. 329–338 (2005).
19. 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 Int. Conf. on Intelligent Robots and Systems*, Beijing, pp. 4946–4952 (2006).
20. N. Mavridis and D. Roy, Grounded situation models for robots: where words and percepts meet, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Beijing, p. 3 (2006).
21. S. Flake, W. Müller and J. Ruf, Structured English for model checking specification, in: *Proc. GI-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Berlin, pp. 2547–2552 (2000).

22. D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi and G. J. Pappas, Valet parking without a valet, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, San Diego, CA, pp. 572–577 (2007).
23. G. E. Fainekos, A. Girard and G. J. Pappas, Hierarchical synthesis of hybrid controllers from temporal logic specifications, in: *Hybrid Systems: Computation and Control* (LNCS 4416), Springer, Berlin, pp. 203–216 (2007).

About the Authors

Hadas Kress-Gazit graduated with a BS in Electrical Engineering from the Technion, in 2002. During her undergraduate studies she worked as a Hardware Verification Engineer for IBM. After graduating and prior to entering graduate school she worked as an Engineer for RAFAEL. In 2005, she received the MS in Electrical Engineering from the University of Pennsylvania, where she is currently pursuing a PhD. Her research focuses on generating robot controllers that satisfy high-level tasks using tools from the formal methods, hybrid systems and computational linguistics communities. She was a finalist for the Best Student Paper Award at ICRA 2007 and a finalist for the Best Paper award at IROS 2007.

Georgios E. Fainekos is currently a Doctoral candidate in the Department of Computer and Information Science at the University of Pennsylvania under the supervision of Professor George J. Pappas. He received his Diploma degree in Mechanical Engineering from the National Technical University of Athens, in 2001, and his MS degree in Computer and Information Science from the University of Pennsylvania, in 2004. His research interests include formal methods, hybrid and embedded control systems, real-time systems, robotics, and unmanned aerial vehicles. He was finalist for the Best Student Paper Award in ICRA 2007.

George J. Pappas received the PhD degree in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1998. He is currently a Professor in the Department of Electrical and Systems Engineering, and the Deputy Dean of the School of Engineering and Applied Science. He also holds secondary appointments in the Departments of Computer and Information Sciences, and Mechanical Engineering and Applied Mechanics. His research focuses on the areas of hybrid and embedded systems, hierarchical control systems, distributed control systems, nonlinear control systems, and geometric control theory, with applications to robotics, unmanned aerial vehicles and biomolecular networks. He coedited *Hybrid Systems: Computation and Control* (Springer, 2004). He was the recipient of a National Science Foundation (NSF) Career Award in 2002, as well as the 2002 NSF Presidential Early Career Award for Scientists and Engineers. He received the 1999 Eliahu Jury Award for Excellence in Systems Research from the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His and his students' papers were finalists for the Best Student Paper Award at the IEEE Conference on Decision and Control (1998, 2001, 2004, 2006), the American Control Conference (2001 and 2004), and ICRA (2007).