High-level Behavior Synthesis for an ATLAS Humanoid Robot

Spyros Maniatopoulos*, Philipp Schillinger†, Vitchyr H. Pong*, David C. Conner‡, and Hadas Kress-Gazit*

***Abstract*— In this paper we ...**

Body of my TODO example

LIST OF TODOS, FIXES, OPEN ISSUES

This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Lab (AFRL) contract FA8750-12-C-0337.

*Verifiable Robotics Research Group, Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA {sm2296, vhp22, hadaskg}@cornell.edu

†Robert Bosch GmbH, Corporate Research, Department for Cognitive Systems, 70442 Stuttgart, Germany philipp.schillinger@de.bosch.com

‡Capable Humanitarian Robotics & Intelligent Systems Lab, Department of Physics, Computer Science and Engineering, Christopher Newport University, Newport News, VA 23606, USA david.conner@cnu.edu

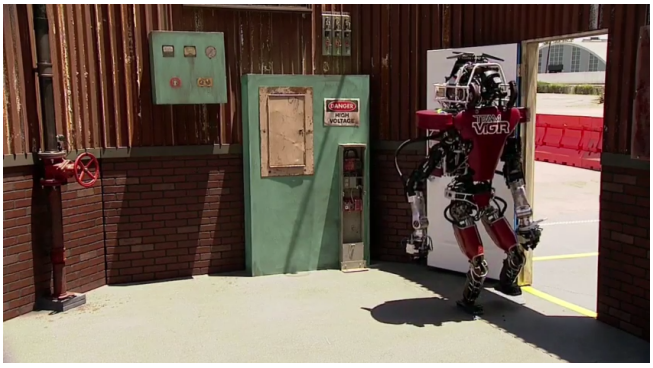


Fig. 1: Team ViGIR's ATLAS humanoid robot on the first day of the DRC Finals. (Photo credit: DARPA)

I. INTRODUCTION

...

Example 1: Consider the high-level task: "Walk to the valve and turn it" (Fig. 1). This would be an intuitive way to express the task from a non-expert user's point-of-view. However, ...(not formal, preconditions, templates, reaction to failures, etc.)

Contributions (brain dump):

- Partial to full specification
  - Most intuitive from the users point-of-view
  - Limited message size over bad comms (send partial specification → compile and synthesize onboard)
- Multi-paradigm specification (objectives and initial conditions from user, topology/modes, preconditions, task)

Also consider mutex for grounding conflicts?

- Generalization of activation–completion paradigm [1]
- Integration with FlexBE and ROS
- Experimental validation on ATLAS

Literature Survey:

- Vasu's "fast-slow" paradigm [1]
- Alternatives to GR(1) Synthesis
  - STRIPS-type planners
  - Optimization-based LTL synthesis (Eric Wolff et al.)
  - **co-safe LTL** (Lydia Kavraki, Calin Belta, etc.)

II. PRELIMINARIES

A. ATLAS *Humanoid Robot*

ATLAS (Fig. 1) is an anthropomorphic robot developed by Boston Dynamics, Inc. (BDI). Team ViGIR chose to leverage the basic capabilities provided by the Boston Dynamics

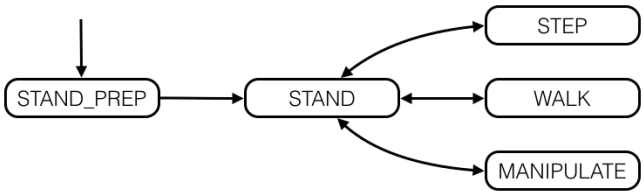


Fig. 2: Excerpt from the BDI control mode interface. Some changes between modes (depicted as arrows) are unidirectional while others are bidirectional. After Team ViGIR's initial checkout, ATLAS is in the STAND_PREP control mode.

Application Programming Interface (API). In the context of this work, we are especially interested in BDI's "control mode" interface (Fig. 2). The active control mode dictates which joints are controlled by the low-level BDI controllers and which joints we can command. For example, in STAND and MANIPULATE BDI's software handles balancing.

ATLAS is equipped with a number of sensors, most notably a Carnegie Robotics Multisense SL[1] mounted as the head. For the DRC Finals, ATLAS was equipped with two Robotiq 3-finger hands,[2] providing manipulation capabilities.

B. Team ViGIR's Approach to the DRC Finals

Team ViGIR based its software on the Robot Operating System (ROS) [2], [3]. In this section, we highlight some elements of the software's design that are relevant to high-level control and behavior synthesis. For a complete overview of Team ViGIR's approach, we refer the interested reader to [4]. From this point on, we will refer to ATLAS running Team ViGIR's software as the *system*.

If the new JFR paper is accepted by February, replace this old reference, [4], in the final submission.

As mentioned, Team ViGIR uses BDI's control mode interface. This means that some system capabilities are preconditioned on a certain control mode being active. For example, in order to execute an arm trajectory, the system must be in MANIPULATE. In addition, the system's operation has to respect the constraints on the possible control mode changes (Fig. 2).

In terms of manipulation, Team ViGIR employs the concept of object templates [5]. In brief, the system presents the human operator with perception data, e.g., a point cloud. Then, the operator detects objects of interest and overlays an object template on top of them. These templates contain metadata, such as relative robot poses from which the object is reachable, relative pre-grasp and grasp end-effector poses, as well as finger configurations corresponding to different grasps. In addition, object templates provide manipulation affordances. For instance, the "door" template provides affordances such as "turn (handle) clockwise" and "push".

Skim Albert's paper and improve this description of OT.

[1] http://carnegierobotics.com/multisense-sl
[2] http://robotiq.com/products/industrial-robot-hand

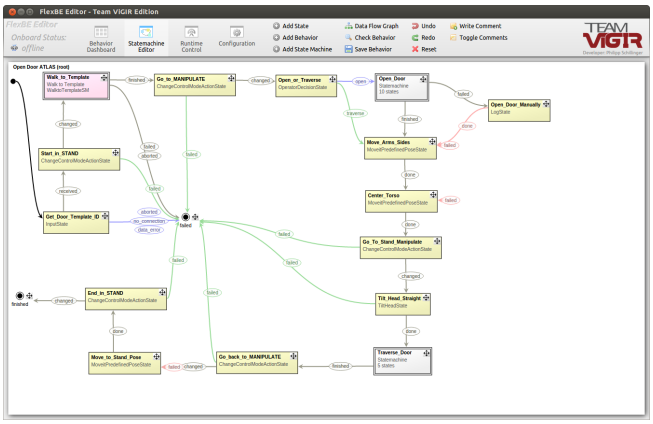


Fig. 3: A manually designed high-level behavior for carrying out the DRC Finals' "Door" task. The initial state is indicated by the black arrow originating from the top left. The behavior has two outcomes, "finished" (bottom left) and "failed" (center). Yellow states are parametrized state implementations, gray states are state machines, and purple states are other high-level behaviors embedded in this one.

High-level Control: Team ViGIR's approach to high-level control is especially relevant to this work. Its corner stone is the Flexible Behavior Engine[3] (FlexBE) [7], [8], which is a major extension of the SMACH high-level executive [6].

Based on the FlexBE framework, developers create "state implementations", $s \in S$. These are small, atomic blocks of code that each interact with some primitive, lower-level, system functionality, $p \in P$. Furthermore, each state implementation defines a number of outcomes $Out(s)$, e.g. "done", "failed", "aborted". The state implementations can be composed to form hierarchical state machines (SM), which encode the logic of execution as well as the flow of data. State machines also have outcomes themselves. The top-level state machine will be referred to as a "behavior".[4] All states of a state machine, S_{SM}, are parametrized instantiations of the state implementations. The composition of new state machines is done manually by an expert user.

Finally, development of behaviors and supervision of execution takes place in FlexBE's graphical user interface[5] (GUI). Figure 3 depicts an example of a high-level behavior designed manually in FlexBE GUI's Editor.

C. Linear Temporal Logic and Reactive LTL Synthesis

...Atomic propositions, LTL, environment vs system, etc.

GR(1) formulas φ have an assume-guarantee structure between the environment (e) and the system (s):

$$\begin{aligned} \varphi &= (\varphi_e \Rightarrow \varphi_s), \\ \varphi_e &= \varphi_e^i \wedge \varphi_e^t \wedge \varphi_e^g, \\ \varphi_s &= \varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g, \end{aligned} \tag{1}$$

[3] https://github.com/team-vigir/flexbe_behavior_engine
[4] https://github.com/team-vigir/vigir_behaviors
[5] https://github.com/team-vigir/flexbe_chrome_app

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

GR(1) Synthesis [10] ... The result is a finite-state automaton; specifically, a Mealy machine.

If there's any space left, define the FSA/Mealy machine mathematically. Possibly connect to FlexBE SMs.

III. Problem Statement

Problem 1 (Discrete Abstraction): Given ATLAS' control mode transition constraints and the available actions $\mathcal{A}$, define a discrete abstraction $\mathcal{D}$ of the robot-plus-software system, S, that captures the execution and outcomes of the atomic capabilities (control mode transitions and actions). In addition, maintain a mapping, $\mathcal{D}_M : S \to AP$, between the atomic capabilities of the system S and the corresponding elements of the discrete abstraction, $\pi \in AP$.

In practice, the mapping $\mathcal{D}_M$ is not one-to-one. But we can treat it as one-to-one to simplify things for the paper.

In general, we would have included the robot's workspace in $\mathcal{D}$. However, that was not necessary in the context of the DRC. Mention in footnote?

Problem 2 (Formal Task Specification): Given a task in terms of goals $\mathcal{G}$, the task's initial conditions $\mathcal{I}$, and the discrete abstraction $\mathcal{D}$ of the system S that is to carry out the task, automatically generate a specification $\mathcal{T}_S$ that encodes, in a formal language, the task being carried out by S.

Problem 3 (Behavior Synthesis): Given a formal task specification, $\mathcal{T}_S$, and the mapping $\mathcal{D}_M$, automatically generate a software implementation of a discrete, high-level, control strategy that is verifiably guaranteed to satisfy $\mathcal{T}_S$.

Revisiting Example 1, we could say that the task's goals $\mathcal{G}$ are $\{\texttt{approach\_valve}, \texttt{turn\_valve}\}$ and the initial conditions $\mathcal{I}$ are, e.g., $\{\texttt{STAND}\}$. ...

IV. Discrete Abstraction

A. Control Modes & Actions

We model ATLAS' control mode interface (c.f. Fig. 2) as a transition system $(\mathcal{M}, \rightarrow)$, where $\mathcal{M}$ is the set of states, each corresponding to one control mode, $m \in \mathcal{M}$, and $\rightarrow$ is a set of valid control mode transitions (subset of $\mathcal{M} \times \mathcal{M}$). In addition, we define $Adj(m) = \{m' \in \mathcal{M} \mid (m, m') \in \rightarrow\}$ and also allow self-transitions, i.e., $m \in Adj(m),\ \forall m \in \mathcal{M}$.

Furthermore, ATLAS can perform actions. Each action, $a \in \mathcal{A}$, corresponds to an atomic capability of the system, e.g., generation of a footstep plan or closing the robot's fingers. Actions may also have one or more preconditions. Action preconditions can be control modes or other actions, i.e., $Prec(a) \in 2^{(\mathcal{A} \cup \mathcal{M})},\ a \notin Prec(a),\ \forall a \in \mathcal{A}$.

B. Atomic Propositions

Write LTL formulas in terms of any possible outcome, $o \in Out(a)$, or only of completion and failure, $\{c, f\}$?

We adopt a paradigm that generalizes the one in [1]. We abstract the discrete actions, $a \in \mathcal{A}$, that ATLAS can perform using one system proposition, π_a, per action and one environment proposition, π_a^o, per possible outcome of that action, $o \in Out(a)$. Similarly,[6] for each control mode, $m \in \mathcal{M}$, we have a system proposition π_m and a number of outcome propositions π_m^o. For both actions and control mode transitions, the outcomes that are of most interest in the context of this paper are completion (c) and failure (f) of the action. That is, $Out(a) = Out(m) = \{c, f\}$ Therefore, the set of atomic propositions AP is given by Eq. (2):

$$\mathcal{Y} = \bigcup_{a \in \mathcal{A}} \pi_a \bigcup_{m \in \mathcal{M}} \pi_m, \tag{2a}$$

$$\mathcal{X}' = \mathcal{X} \bigcup_{a \in \mathcal{A}} \bigcup_{o \in Out(a)} \pi_a^o \bigcup_{m \in \mathcal{M}} \bigcup_{o \in Out(m)} \pi_m^o, \tag{2b}$$

where $\mathcal{X}$ are environment propositions other than outcome propositions, e.g., ones that abstract sensors, as per [9].

C. Proposition Grounding / Mapping

Now that we have props, show what $\mathcal{D}_M$ looks like.

Comment on the possibility of mapping multiple outcomes to one proposition, since it appears in Fig. 6a

V. Formal Task Specification

A. Multi-Paradigm Specification

Specifying a robot task in a formal language can be time consuming and error prone. It also requires an expert user. To alleviate these issues, we employ a multi-paradigm specification approach. We first observe that there are portions of the task specification $\mathcal{T}_S$ that are going to be system-specific and portions that are going to be task-specific, such as the task's goals. Intuitively, a user should only have to specify the goals without worrying about the internals of the robot and the software it is running. For ATLAS, we can automatically infer which control modes and actions are pertinent to a task. Finally, the initial conditions are either specified by the user or detected at runtime.

Referring to Problem 2, we get the goals, $\mathcal{G}$, and initial conditions, $\mathcal{I}$, from the user. The discrete abstraction, $\mathcal{D}$, is ATLAS-specific and we assume that it has been created a priori from expert developers, according to Section IV. We can now automatically compile the task specification $\mathcal{T}_S$ in (the GR(1) fragment [10] of) Linear Temporal Logic. Since LTL is compositional, we can generate individual LTL formulas and then conjunct them to get the full LTL specification.

B. Specification of Actions and Control Mode Constraints

1) Generic Formulas: We say that an activation proposition π_p, $p \in \{a, m\}$, is `True` when the corresponding primitive functionality is being activated and `False` when it is not

[6] The distinction between action and control mode propositions is purely for the sake of clarity of notation. There is nothing special about either.

being activated[7]. Therefore, the system safety requirement (3) dictates that all activation propositions $\pi_p \in \mathcal{Y}$ should turn `False` once an outcome has been returned.

$$\bigwedge_{o \in Out(p)} \square \left(\pi_p \wedge \bigcirc \pi_p^o \Rightarrow \bigcirc \neg \pi_p \right) \tag{3}$$

The environment safety assumption (4) dictates that the outcomes, π_p^o, of the activation of any primitive functionality p are mutually exclusive. For example, an action cannot both succeed and fail. Formula (4) also allows for no outcome being `True`.

Formula (4) requires the $\bigcirc$ operators to synthesize properly (slugs), but intuitively, they shouldn't be there.

$$\bigwedge_{o \in Out(p)} \square \left(\bigcirc \pi_p^o \Rightarrow \bigwedge_{o' \neq o} \bigcirc \neg \pi_p^{o'} \right) \tag{4}$$

The environment safety assumption (5) constraints the value of outcomes. Specifically, it dictates that, if an outcome is `False` and the corresponding primitive functionality is not being activated, then that outcome should remain `False`. It is a generalization of formula (4) in [1].

$$\bigwedge_{o \in Out(p)} \square \left(\neg \pi_p^o \wedge \neg \pi_p \Rightarrow \bigcirc \neg \pi_p^o \right) \tag{5}$$

2) Action-specific Formulas: The following formulas encode the connection between the activation and the possible outcomes of the robot's actions, $a \in \mathcal{A}$.

The environment safety assumptions (6) dictate that the value of an outcome should not change if the corresponding action has not been activated again. In other words, outcomes persist.

$$\bigwedge_{o \in Out(a)} \square \left(\pi_a^o \wedge \neg \pi_a \Rightarrow \bigcirc \pi_a^o \right) \tag{6}$$

The environment liveness assumption (7) is a fairness condition. It states that (always) eventually, the activation of an action will result in an outcome. The disjunct $\neg \pi_a$ is added in order to prevent the system from winning the game by never activating the action.

$$\square \lozenge \left(\left(\pi_a \wedge \bigvee \bigcirc \pi_a^o \right) \vee \neg \pi_a \right) \tag{7}$$

The system safety requirement (8) demonstrates how a formula encoding the preconditions of an action, $Prec(a)$, looks like in the activation–outcomes paradigm.

Demonstrate how, given partial specification, we can bring in only those actions and modes that are necessary.

$$\square \left(\bigvee_{p \in Prec(a)} \neg \pi_p^c \Rightarrow \neg \pi_a \right) \tag{8}$$

where the superscript $c \in Out(p)$ stands for "completion".

[7]Note that this is in contrast to [1], where π_p being `False` stands for the primitive functionality p being *deactivated*, e.g., turning a camera off.

3) Control Mode Formulas: For brevity of notation, let

$$\varphi_m = \pi_m \wedge \bigwedge_{m' \neq m} \neg \pi_{m'}$$

Activating φ_m, as opposed to π_m, takes into account the mutual exclusion between control modes $m \in \mathcal{M}$. Also let

$$\varphi_{\mathcal{M}}^{none} = \bigwedge_{m \in \mathcal{M}} \neg \pi_m,$$

where $\varphi_{\mathcal{M}}^{none}$ being `True` stands for not activating any control mode transitions, i.e., staying in the same control mode.

The system safety requirements (9) encode a topological transition relation, such as the BDI control mode transition system.

$$\bigwedge_{m \in \mathcal{M}} \square \left(\bigcirc \pi_m^c \Rightarrow \bigvee_{m' \in Adj(m)} \bigcirc \varphi_{m'} \vee \bigcirc \varphi_{\mathcal{M}}^{none} \right) \tag{9}$$

The environment safety assumptions (10) enforce mutual exclusion between the BDI control modes.

Formula (10) also requires the $\bigcirc$ operators to synthesize properly (slugs), but intuitively, they shouldn't be there.

$$\bigwedge_{m \in \mathcal{M}} \square \left(\bigcirc \pi_m^c \Leftrightarrow \bigwedge_{m' \neq m} \bigcirc \neg \pi_{m'}^c \right) \tag{10}$$

The environment safety assumptions (11) govern how the active control mode can change in a single time step in response to the activation of a control mode transition.

$$\bigwedge_{m \in \mathcal{M}} \bigwedge_{m' \in Adj(m)} \square \left(\pi_m^c \wedge \varphi_{m'} \Rightarrow \left(\bigcirc \pi_m^c \bigvee_{o \in Out(m')} \bigcirc \pi_{m'}^o \right) \right) \tag{11}$$

The environment safety assumptions (12) dictate that the value of the outcomes of control mode transitions must not change if no transition is being activated, i.e., they must persist.

$$\bigwedge_{m \in \mathcal{M}} \bigwedge_{o \in Out(m)} \square \left(\pi_m^o \wedge \varphi_{\mathcal{M}}^{none} \Rightarrow \bigcirc \pi_m^o \right) \tag{12}$$

The environment liveness assumption (13) is the equivalent of the fairness condition (7) for control mode transitions. This single formula accounts for all control modes.

$$\square \lozenge \left(\bigvee_{m \in \mathcal{M}} \left(\varphi_m \wedge \bigvee_{o \in Out(m)} \bigcirc \pi_m^o \right) \vee \varphi_{\mathcal{M}}^{none} \right) \tag{13}$$

C. Specification of Initial Conditions

For each action, a, and control mode, m, in the initial conditions, $\mathcal{I}$, the completion proposition should be `True` in the environment initial conditions (14). All other outcome propositions corresponding to those actions and control modes, as well as all outcome propositions corresponding to any other actions and control modes, should be `False`.

$$\varphi_i^e = \bigwedge_{i \in \mathcal{I}} \left(\pi_i^c \bigwedge_{o \in Out(i) \setminus \{c\}} \neg \pi_i^o \right) \wedge \bigwedge_{j \notin \mathcal{I}} \bigwedge_{o \in Out(j)} \neg \pi_j^o \tag{14}$$

Activation propositions are `False` regardless of whether that action or control mode is in the initial conditions or not (15). The reason being that, intuitively, if we want something to be an initial condition, then we shouldn't have the resulting controller re-activate it at the beginning of execution.

$$\varphi_i^s = \bigwedge_{i \in \mathcal{I}} \neg \pi_i \wedge \bigwedge_{j \notin \mathcal{I}} \neg \pi_j \tag{15}$$

Do I move ICs to the end of the section and include memory props explicitly? Or leave them here, but state memory ICs along with "infinite-to-finite" formulas?

D. Specification of Task Goals

The system initial condition (16), safety requirements (17) and (18), and liveness requirement (19) are used to reason about the satisfaction of the system's goals, $g \in \mathcal{G}$, in a finite run (as opposed to infinite execution, which is what LTL is defined over). In this finite run paradigm, the synthesized state machine (SM) itself has outcomes, $o \in Out(SM)$. The propositions corresponding to the SM's outcomes, π^o_{SM}, are system, not environment, propositions. The system propositions, μ_g, serve as memory of having accomplished each goal (c.f. [11]).

$$\bigwedge_{g \in \mathcal{G}} \neg \mu_g \tag{16}$$

$$\bigwedge_{g \in \mathcal{G}} \square \left(\bigcirc \pi^c_g \vee \mu_g \Leftrightarrow \bigcirc \mu_g \right) \tag{17}$$

$$\square \left(\pi^c_{SM} \Leftrightarrow \bigwedge_{g \in \mathcal{G}} \mu_g \right) \tag{18a}$$

$$\square \left(\pi^f_{SM} \Leftrightarrow \bigvee_{\pi \in \mathcal{Y}} \pi^f \right) \tag{18b}$$

$$\bigwedge_{o \in Out(SM)} \square \left(\pi^o_{SM} \Rightarrow \bigcirc \pi^o_{SM} \right) \tag{18c}$$

$$\square \diamondsuit \left(\bigvee_{o \in Out(SM)} \pi^o_{SM} \right) \tag{19}$$

The time complexity of synthesis is cubic in the number of liveness requirements. We save by only having one. Although that's probably dominated by the complexity being exponential in the number of propositions.

Formula (17) does not guarantee that the goals will be achieved in a specific order. However, that is often desirable. To this end, we can define the goals as an ordered set $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, where $g_i < g_j$ for $i < j$, and the relation $g_i < g_j$ means that goal g_i has to be achieved before g_j. With this definition, we can replace the safety requirement (17) with (20), whenever strict goal order is desired.

$$\bigwedge_{i=1}^{n} \square \left((\pi_{g_i} \wedge \bigcirc \pi^c_{g_i}) \wedge \mu_{g_{i-1}} \vee \mu_{g_i} \Leftrightarrow \bigcirc \mu_{g_i} \right), \tag{20}$$

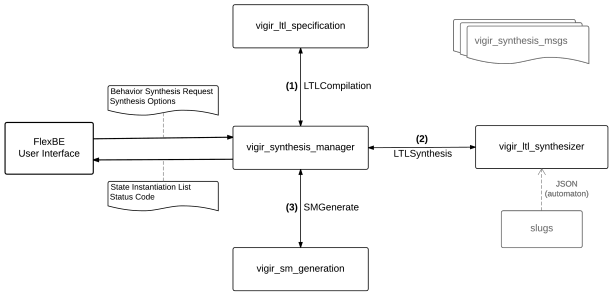


Fig. 4: Team ViGIR's "Behavior Synthesis" ROS packages and the nominal workflow (clockwise, starting from the left).

(i) Add ReSpeC to image (move msgs to the left) (ii) Either number all steps or none (iii) Rename 3 services.

where $\mu_{g_0} \triangleq$ `True`. Formula (20) forces the system to carry out goal g_i after it has accomplished goal g_{i-1}. It can still activate the action corresponding to π_{g_i} earlier, as necessitated by other parts of the task, but that will not count towards achievement of g_i, as indicated by μ_{g_i} being `True`.

Finally, these auxiliary (memory and SM outcome) propositions have to be added to the system propositions:

$$\mathcal{Y}' = \mathcal{Y} \bigcup_{g \in \mathcal{G}} \mu_g \bigcup_{o \in Out(SM)} \pi^o_{SM}$$

VI. High-level Behavior Synthesis

We tackle Problem 3 in two sequential steps. First, we automatically generate a correct-by-construction automaton from the formal specification $\mathcal{T}_S$ using GR(1) synthesis (see [10] and Section II-C). Specifically, we employ the synthesis algorithm in [12], which can handle a slightly larger fragment of LTL than GR(1). Namely, the one that includes $\bigcirc$ (next) operators in liveness formulas, such as in formulas (7) and (13). This algorithm was first used in [1].

@HKG, is it true that Vasu's paper was the first case of synthesis for this fragment? Had Ruediger used it before?

Second, we use the mapping $\mathcal{D}_M^{-1} : AP \to S$ to instantiate the abstract automaton as a concrete software implementation, i.e., an executable state machine in the SMACH/FlexBE framework (see Section II-B).

VII. ROS Implementation

We have implemented all aspects of our approach in `vigir_behavior_synthesis`,[8] a collection of Robot Operating System (ROS) Python packages. Figure 4 depicts these packages as well as the nominal workflow.

The synthesis action server (`vigir_synthesis_manager`) receives a request from the user via FlexBE's GUI. Given the user's input (initial conditions and goals), the server first requests a full set of LTL formulas for ATLAS from the `GenerateLTLSpecification` service (`vigir_ltl_specification` package). The generation of

[8] https://github.com/team-vigir/vigir_behavior_synthesis

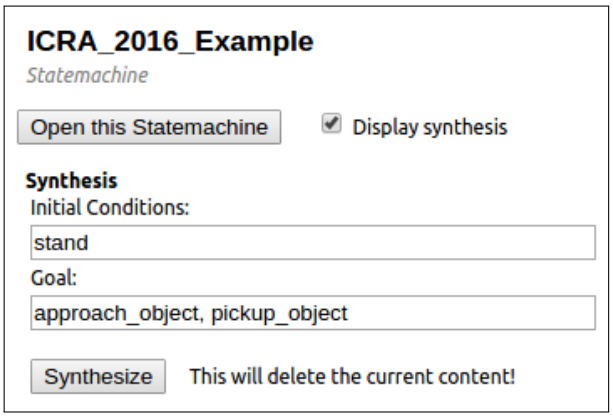


Fig. 5: Screenshot of the FlexBE Editor's synthesis menu.

the LTL formulas from Section V is delegated to our "Reactive Specification Construction kit" (ReSpeC),[9] which is a Python framework with rudimentary ROS integration.

The `vigir_ltl_synthesizer` package acts as a wrapper for external synthesis tools (currently, [12] is supported). Given the generated LTL specification, the `SynthesizeAutomaton` service returns a finite-state automaton that is guaranteed to satisfy it, if one exists. Finally, the server requests a `StateInstantiation` message from the `GenerateFlexBESM` service (`vigir_sm_generation` package). This message provides the FlexBE Editor with sufficient information to generate Python code, i.e., an executable state machine that instantiates the synthesized automaton. The corresponding action, services, and messages are defined in the `vigir_synthesis_msgs` package.

The following excerpt[10] is from the `StateInstantiation` list message, which is the end product of the `vigir_behavior_synthesis` workflow (see Fig. 4). Specifically, this excerpt corresponds to the primitive functionality `object_template`, which appears in Fig. 6a.

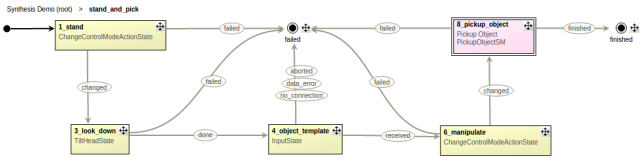
```
state_path: /4_object_template
state_class: InputState
parameter_names: [request]
parameter_values: [InputState.SELECTED_OBJECT_ID]
outcomes: [no_connection, aborted, received, data_error]
transitions: [failed, failed, 6_manipulate, failed]
autonomy: [0]
userdata_keys: [data]
userdata_remapping: [template_id]
```

VIII. Experimental Validation

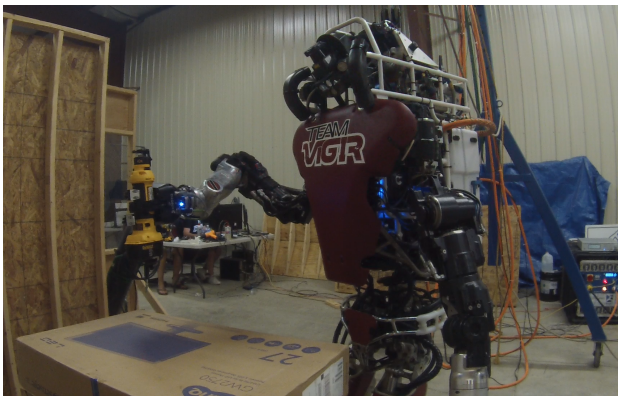
Team ViGIR did not employ high-level behavior synthesis during the DRC Finals. However, we later carried out experimental demonstrations on ATLAS in the lab. Due to a hardware issue, ATLAS could not locomote. Thus, in addition to two experimental demonstrations, we present a simulation run carried out in Gazebo, using the same operator and onboard software. We summarize these demonstrations below. Please also refer to the accompanying video.

[9] https://github.com/team-vigir/ReSpeC

[10] We have omitted some details for the sake of brevity and clarity of presentation. For example, most list elements are strings, e.g., `"template_id"`.



(a) The state machine above was synthesized for the task with $\mathcal{I} = \{\texttt{stand\_prep}\}$ and $\mathcal{G} = \{\texttt{look\_down}, \texttt{pickup\_object}\}$.



(b) ATLAS finishing execution of state `8_pickup_object`.

Fig. 6: These correspond to the first part of the video attachment.

A. Behavior Development using Synthesis

In the first experimental demo, we show how a high-level behavior is specified and synthesized starting from scratch[11]. Once the state machine has been instantiated (Fig. 6a), it is ready for execution (Fig. 6b).

B. Online Modifications using Synthesis

For the second experimental demonstration, consider a scenario where the operator has designed a state machine that addresses a high-level task (either manually or via synthesis). ATLAS is then deployed and starts carrying out this task. If, during execution, an *unexpected* situation arises, the operator can use FlexBE's runtime modification capability (Fig. 7). In this case, behavior execution is "locked" (paused) at some state (Fig. 7a) and the operator specifies a new high-level behavior meant to address the unexpected situation. Once this new state machine is instantiated (Fig. 7b), it is connected to the previous one (Fig. 7a), and execution resumes.

C. Infinite execution or Footstep stuff (simulation)

For example, "From stand-prep, walk to object, pick it up, (go to stand-manipulate), carry object, release, and repeat"

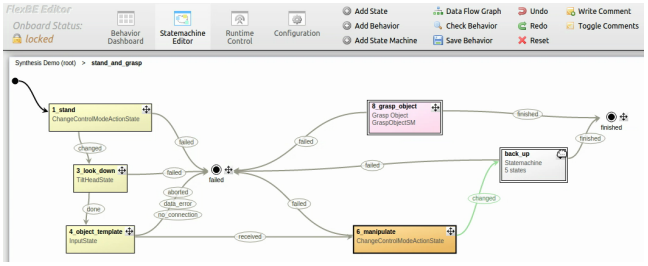
Provide data on how computationally costly/cheap behavior synthesis is. Time vs number of actions?

IX. Conclusions and Future Work

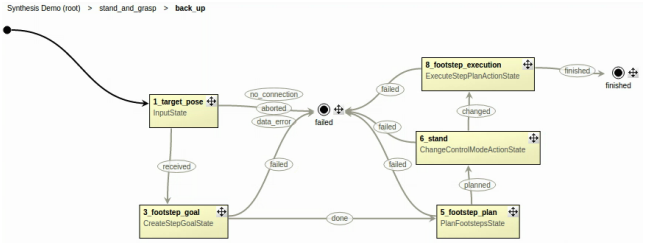
...

...Once we have a synthesized SM, it can be treated as a primitive action with its outcomes, etc.

[11] The generated LTL specification and synthesized automaton are available at: https://gist.github.com/spmaniato/c37fb12e874c73d986da



(a) The operator "locks" the initial state machine at the state `6_manipulate` (indicated by the orange color), which is allowed to be executed. Then, a new state machine, `back_up`, is synthesized with `manipulate` as the initial condition. The transition from `6_manipulate` is then moved from `8_grasp_object` to `back_up`.



(b) The new state machine, `back_up`, was synthesized for the task with $\mathcal{I} = \{\texttt{manipulate}\}$ and $\mathcal{G} = \{\texttt{footstep\_execution}\}$.

Fig. 7: During execution of the `stand_and_grasp` state machine, an unexpected situation arises. In response, the operator pauses execution and specifies a high-level behavior to have ATLAS back away from the object (7b). These snapshots correspond to the second part of the video attachment.

Hint at the question: "What does it mean to offer formal guarantees when the activation of primitives can result in failure?"

…Future work: Capability specification, integrate more robots, …

ACKNOWLEDGMENTS

The authors thank all other members of Team ViGIR and especially Alberto Romay, Stefan Kohlbrecher, and Prof. Oskar von Stryk from Technische Universität Darmstadt.

REFERENCES

[1] V. Raman, N. Piterman, and H. Kress-Gazit, "Provably Correct Continuous Control for High-Level Robot Behaviors with Actions of Arbitrary Execution Durations," in *IEEE Int'l. Conf. on Robotics and Automation*, 2013.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[3] Robot Operating System (ROS). [Online]. Available: www.ros.org

[4] S. Kohlbrecher, A. Romay, A. Stumpf, A. Gupta, O. von Stryk, F. Bacim, D. Bowman, A. Goins, R. Balasubramanian, and D. Conner, "Human-Robot Teaming for Rescue Missions: Team ViGIR's Approach to the 2013 DARPA Robotics Challenge Trials," *Journal of Field Robotics*, vol. 32, no. 3, pp. 352–377, 2015.

[5] A. Romay, S. Kohlbrecher, D. Conner, A. Stumpf, and O. von Stryk, "Template-Based Manipulation in Unstructured Environments for Supervised Semi-Autonomous Humanoid Robots," in *Proc. IEEE-RAS Intl. Conf. Humanoid Robots*, Madrid, Spain, Nov 2014, pp. 979–986.

[6] J. Bohren and S. Cousins, "The SMACH High-Level Executive [ROS News]," *Robotics Automation Magazine, IEEE*, vol. 17, no. 4, pp. 18–20, Dec 2010.

[7] P. Schillinger, "Development of an Operator Centric Behavior Control Approach for a Humanoid Robot," Master's thesis, Technische Universität Darmstadt, 2013.

[8] ——, "An Approach for Runtime-Modifiable Behavior Control of Humanoid Rescue Robots," Master's thesis, Technische Universität Darmstadt, 2015.

[9] 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.

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

[11] V. Raman, B. Xu, and H. Kress-Gazit, "Avoiding forgetfulness: Structured English specifications for high-level robot control with implicit memory," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2012.

[12] R. Ehlers, V. Raman, and C. Finucane. (2013–2015) Slugs GR(1) synthesizer. [Online]. Available: https://github.com/VerifiableRobotics/slugs