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 DARPA / TORC Robotics ...

*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 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.

Each author should review and fix their contact info.

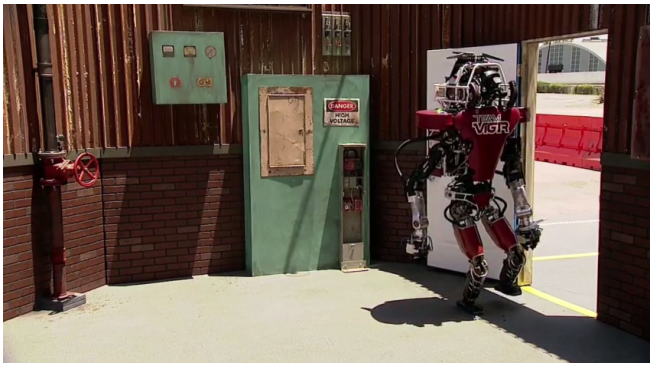

Fig. 1: Team ViGIR's ATLAS humanoid robot on the first day of the DRC Finals. (Photo credit: DARPA)

I. INTRODUCTION

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:

- Alternatives to GR(1) Synthesis
  - co-safe LTL (Lydia Kavraki, Calin Belta, etc.)

II. PRELIMINARIES

A. ATLAS Humanoid Robot

...

B. Team ViGIR's Approach to High-level Control

...FlexBE[1] , GUI[2] , behaviors and states[3].

C. Linear Temporal Logic and Reactive LTL Synthesis

...

[1] https://github.com/team-vigir/flexbe_behavior_engine
[2] https://github.com/team-vigir/flexbe_chrome_app
[3] https://github.com/team-vigir/vigir_behaviors

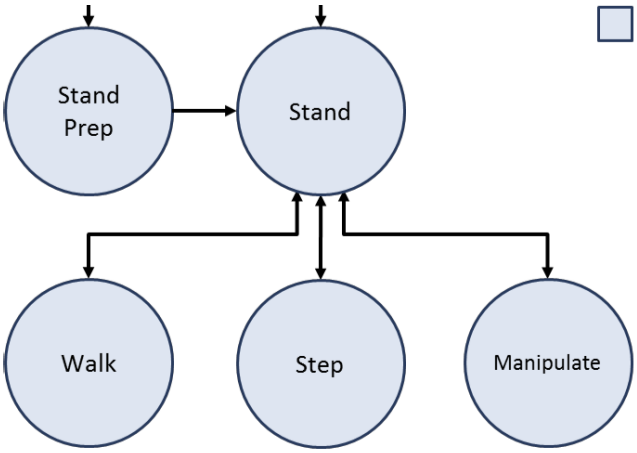

Fig. 2:

Placeholder! Create simple control mode TS figure (flat).

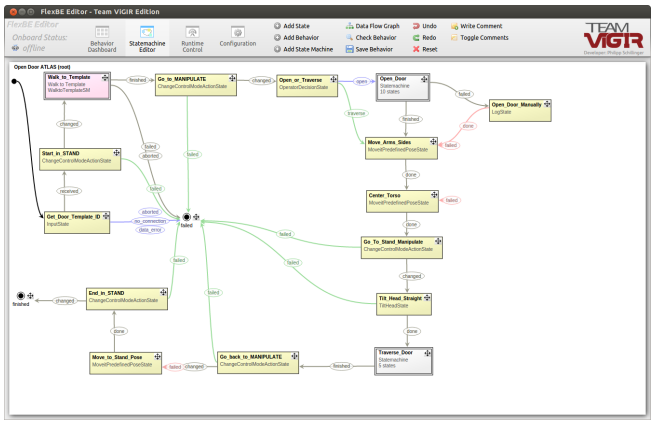

Fig. 3: A high-level behavior for carrying out the DRC Finals' "Door" task. It was designed by Team ViGIR developers using the FlexBE Editor (graphical user interface).

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 software implementation of control mode transitions and actions. In addition, maintain a mapping, $\mathcal{D}_M : S \to \mathcal{AP}$, between the elements of the discrete abstraction and the corresponding software components.

In practice, the mapping $\mathcal{D}_M$ is one-to-many. But we can treat it as one-to-one to simplify the 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.

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 compile 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 $\mathcal{T}_S$.

IV. Discrete Abstraction

A. Control Modes & Actions

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

ATLAS can perform actions. Each action, $a \in \mathcal{A}$, is considered discrete and it corresponds to a software component, 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,[4] 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. (1):

$$\mathcal{Y} = \bigcup_{a \in \mathcal{A}} \pi_a \bigcup_{m \in \mathcal{M}} \pi_m, \tag{1a}$$

$$\mathcal{X}' = \mathcal{X} \bigcup_{a \in \mathcal{A}} \bigcup_{o \in Out(a)} \pi_a^o \bigcup_{m \in \mathcal{M}} \bigcup_{o \in Out(m)} \pi_m^o, \tag{1b}$$

where $\mathcal{X}$ are environment propositions other than outcome propositions, e.g., ones that abstract sensors, as per [2].

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

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

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 [3] 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: The system safety requirements (2) dictate that an activation proposition should turn `False` once an outcome has been returned.

$$\bigwedge_{o \in Out(a)} \square \left(\pi_a \wedge \bigcirc \pi_a^o \Rightarrow \bigcirc \neg \pi_a \right) \tag{2}$$

The environment safety assumptions (3) dictate that the outcomes of an action are mutually exclusive. For example, an action cannot both succeed and fail.

Formula (3) requires the $\bigcirc$ operators to synthesize properly (slugs), but intuitively, they shouldn't be there.

$$\bigwedge_{o \in Out(a)} \square \left(\bigcirc \pi_a^o \Rightarrow \bigwedge_{o' \neq o} \bigcirc \neg \pi_a^{o'} \right) \tag{3}$$

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 (4) govern the value of outcomes in the next time step. Specifically, formula (4) dictates that, if an outcome is `False` and the corresponding action is not activated, then that outcome should remain `False`. It is a generalization of formula (4) in [1].

$$\bigwedge_{o \in Out(a)} \square \left(\neg \pi_a^o \wedge \neg \pi_a \Rightarrow \bigcirc \neg \pi_a^o \right) \tag{4}$$

The environment safety assumptions (5) dictate that the value of an outcome should not change if the corresponding action has not been activated again. In other words, outcomes persist.

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

The environment liveness assumption (6) 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.

$$\square \diamondsuit \left(\left(\pi_a \wedge \bigvee \bigcirc \pi_a^o \right) \vee \neg \pi_a \right) \tag{6}$$

The system safety requirement (7) 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.

$$\square \left(\bigvee_{p \in Prec(a)} \neg \pi_p^c \Rightarrow \neg \pi_a \right) \tag{7}$$

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

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 (8) encode a topological transition relation, such as the BDI control mode transition system.

$$\bigwedge_{m \in \mathcal{M}} \square \left(\bigcirc \pi_m^c \Rightarrow \bigvee_{m' \in Adj(m)} \bigcirc \varphi_{m'} \vee \bigcirc \varphi_{\mathcal{M}}^{none} \right) \tag{8}$$

The environment safety assumptions (9) enforce mutual exclusion between the BDI control modes.

Formula (9) also requires the $\bigcirc$ operators to synthesize properly (slugs), but intuitively, they shouldn't be there.

$$\bigwedge_{m \in \mathcal{M}} \square \left(\bigcirc \pi_m^c \Leftrightarrow \bigwedge_{m' \neq m} \bigcirc \neg \pi_{m'}^c \right) \tag{9}$$

The environment safety assumptions (10) govern how the active control mode can change in a single time step in response to the activation of a control mode transition.

$$\bigwedge_{m \in \mathcal{M}} \bigwedge_{m' \in Adj(m)} \square \left(\pi_m^c \wedge \varphi_{m'} \Rightarrow \left(\bigcirc \pi_m^c \bigvee_{o \in Out(m')} \bigcirc \pi_{m'}^o \right) \right) \tag{10}$$

The environment safety assumptions (11) constrain the outcomes control mode transitions.

$$\bigwedge_{m \in \mathcal{M}} \bigwedge_{o \in Out(m)} \square \left(\neg \pi_m^o \wedge \neg \pi_m \Rightarrow \bigcirc \neg \pi_m^o \right) \tag{11}$$

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 (6) for control mode transitions. This single formula accounts for all control modes.

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

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`.

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

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_{SM}^o, are system, not environment, propositions. The system propositions, μ_g, serve as memory of having accomplished each goal (c.f. [4]).

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

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

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

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

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

$$\square\diamondsuit\Big(\bigvee_{o\in Out(SM)}\pi_{SM}^o\Big) \tag{19}$$

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.

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

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:

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

VI. ROS Implementation

We have implemented all aspects of our approach in `vigir_behavior_synthesis`,[5] 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 LTL Compilation service (`vigir_ltl_specification`). The generation of the LTL formulas from Section V is delegated to our "Reactive Specification Construction kit," which is a ROS-independent Python framework[6].

The LTL Synthesis service (`vigir_ltl_synthesizer`) acts as a wrapper for an external synthesis tool (currently, [5] is supported). Upon request, it returns an automaton that is guaranteed to satisfy the LTL specification, if one exists. Finally, the server requests a state instantiation message from the State Machine Generation service (`vigir_sm_generation`). The resulting message contains instructions that FlexBE can use to generate Python code: an executable state machine that instantiates the synthesized automaton. The corresponding action, services, and messages are defined in the `vigir_synthesis_msgs` package.

[5] https://github.com/team-vigir/vigir_behavior_synthesis
[6] https://github.com/team-vigir/ReSpeC

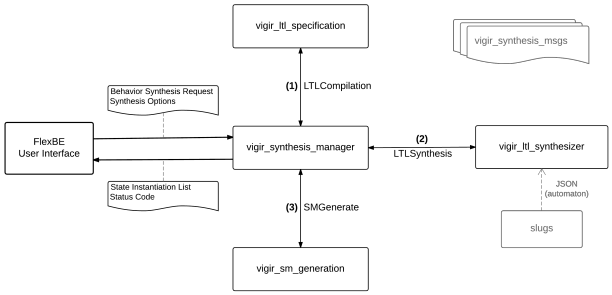

Fig. 4: Team ViGIR's "Behavior Synthesis" ROS packages and the nominal workflow (clockwise, starting from the left).

(i) Add ReSpeC to the image (move msgs to the left) (ii) Either number all arrows or none (iii) If I end up renaming LTLCompilation srv, update image accordingly.

VII. Experimental Validation

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

VIII. Conclusions and Future Work

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

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

[4] 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.

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