A framework for verifying autonomous robotic agents against environment assumptions

Hoang Tung Dinh[0000−0003−2827−9931] and Tom Holvoet[0000−0003−1304−3467]

KU Leuven, Dept. of Computer Science, imec-DistriNet, B-3001 Leuven, Belgium
{hoangtung.dinh, tom.holvoet}@cs.kuleuven.be

Abstract. Guaranteeing safety is crucial for autonomous robotic agents. Formal methods such as model checking show great potential to provide guarantees on agent and multi-agent systems. However, as robotic agents often work in open, dynamic and unstructured environments, achieving high-fidelity environment models is non-trivial. Most verification approaches for agents focus on checking the internal reasoning logic without considering operating environments or focus on a specific type of environments such as grid-based or graph-based environments. In this paper we propose a framework to model and verify the decision making of autonomous robotic agents against assumptions on environments. The framework focuses on making a clear separation between agent modeling and environment modeling, as well as providing formalism to specify agent's decision making and assumptions on environments. As the first demonstration of this ongoing research, we provide an example of using the framework to verify an autonomous UAV agent performing pylon inspection.

Keywords: Verification · Model checking · Robotic.

1 Introduction

Autonomous robotic agents are often considered as safety-critical. They work in an open environment and need to ensure that their behavior does not harm other entities such as human or properties.

Yet, guaranteeing safety for robotic agent's behavior is difficult. As they often work in an open, dynamic, unstructured and uncontrolled environment, ensuring that the correctness of the agent's behavior holds in all possible situations is non-trivial. The complexity of the environment is a major challenge hindering the formal verification of autonomous agents [11], as achieving a formal and high-fidelity environment model to perform verification is difficult, if not impossible.

Due to the complexity in modeling the environment, researches on formal verification of autonomous agents often focus on a specific and low-fidelity representation of environments such as grid-based [1] and graph-based [13], raising concerns on the validity of verification results when deploying agent systems in real-world. Recently, environment representations based on Markov decision processes are also used [10, 9]. Such representations can capture more complicated

behavior of environments but are still not expressive enough to represent, for example, the temporal properties of environments.

In this paper we study the problem of verifying robotic agent's decision making against the specifications of environments. Instead of creating a concrete model of the environment for verification, we verify decision making logics of robotic agents against a set of assumptions, formalized in Linear Temporal Logic (LTL), on the behavior of the environment. The idea of verifying agent's decision making against environment assumptions was introduced in [4]. As having a high-fidelity formal model of the environment is non-trivial while often, not every detail of the environment is relevant to the verification task, it is more realistic and practical to derive a set of necessary assumptions for the agent to guarantee the correctness of its decisions. In addition, from the software engineering perspective, it is desirable to have environment assumptions defined explicitly and formally as they essentially represent the boundary conditions in which the agent system can guarantee its correctness. In constrast, a low-fidelity environment model often consists of many implicit assumptions.

In concrete, our contributions are as follows. We propose a framework to verify the decision making of robotic agents operating in open environments. Our framework provides formalism to represent the discretization logics of the perception information. The environment is represented by a set of LTL assumptions on the perception information before the discretization. We employ the NuXmv model checker that supports various types of systems and model checking techniques, which allows us to represent environments at different levels of fidelity. As the first demonstration of this ongoing research, we provide an example of using the framework to verify an autonomous UAV agent performing pylon inspection.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents a general overview on the architecture of robotic agents. Section 4 describes our proposed framework. Section 5 presents an example of using the framework. Finally, Section 6 draws conclusions and outlines future work.

2 Related work

Formally modeling environments remains one of the most challenging task for verification. Different environment representations were proposed. Aminof et al. [1] propose a framework for verifying multi-agent systems in parameterized grid-environments. Rubin [13] presents a verification framework for mobile agents moving on graphs. In [9, 10], the verification of multi-agent systems is performed with environments modeled as Markov decision processes.

In [5, 7], the behavior of a homecare robot is verified where the environment is modeled as a set of variables representing high-level sensor information such as the fridge door is being open. Those variables can be set non-deterministically at any time. In [17, 16], for the same verification task for a homecare robot, the environment is represented as an agent in Brahms, an agent programming language. The environment model is limited to the non-deterministic choice of

actions of the Brahms agent. Morse et al. [12] represent the environment by a probabilistic model where its actors have uncertainty in their actions' outcomes.

A common limitation of these environment representations is that their expressiveness is limited, making it difficult to encode different assumptions on the behavior of environments. So far, there has been little work focusing on formally specifying environments for model checking. Most close to our work is the work of Dennis et al. [4], where they proposed a verification methodology for agent's decision making in which one can specify environment assumptions as logical formulas on the discretized incoming perceptions of the agent. Our work differs from the work in [4] in that beside the decision making of the agent, we also take into account the discretization logics of the perception information in the verification by providing a formalism to represent the discretization logics. By doing so, we allow environment assumptions to be specified on the information before discretization.

3 Robotic agent architecture

In this section we provide an overview on the architecture of robotic agents. Similar to other types of agents, robotic agents interact with environments via sensing and acting.

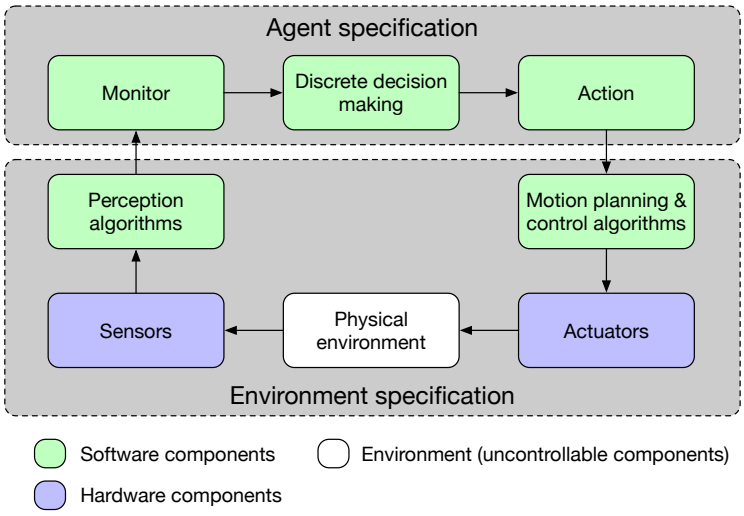

Fig. 1. A typical robotic agent architecture.

Figure 1 illustrates a typical architecture of robotic agents. Robotic agents are cyber physical systems consisting of both software and hardware components. They are equipped with **sensors** that provide raw sensing data about the physical environment. The raw sensing data is then transformed to meaningful

information by **perception algorithms** such as localization and obstacle detection. The outputs of the perception algorithms are often continuous data, for example, the location of the agent in a three dimensional Cartesian coordinate system, or events such as commands from human operators. The perception information is then discretized by **monitors** to symbolic information.

The symbolic information is then taken into account by the **discrete decision making** component. The discrete decision making component performs symbolic reasoning, for example, based on well-known Belief-Desired-Intention (BDI) models or planning techniques [8] to select one or several **actions** to execute. An action could be instantaneous, for example, sending a notification or taking a picture, or it could be durative, for instance, moving to a location. When an action is executed, it activates a set of **motion planning and/or control algorithms** that compute and send commands to the agent's **actuators**, which in turns interact with the physical environment.

To verify the agent's decision making, one needs to specify the agent system and the environment in which the agent operates in. As discussed in Section 1, we specify the environment as a set of logical assumptions. For the verification problem, we need to define the boundary between the agent specification and the environment specification. The boundary will in turn define how the assumptions on the environment can be specified.

Environments are often modeled based on discrete variables provided by the monitors. The effects of agent actions are either specified based on the discrete variables or ignored. It is due to the complexity of the systems. Perception algorithms are often black-box while motion planning and control algorithms often involve complicated optimization processes on continuous domain, making them non-trivial to be formally modeled and verified. Raw sensor data does not have semantic meaning and actuator commands involve the complex dynamic models of physical systems, making them difficult to be formally represented.

Yet, monitors, the software components that discretize the outputs of perception algorithms to symbolic information for decision making, are white-box and can be formally represented. Including monitors in the agent specification is beneficial in two aspects. First, discretization logics in monitors can be formally verified together with the decision making, extending the boundary in which formal guarantees can be provided on the agent system. Second, environment assumptions can be specified on the outputs of perception algorithms, bringing them closer to the actual system implementation and the physical environment.

Because of these reasons, in our proposed framework, the agent specification includes the monitor components, the decision making component and the action components, as shown in Figure 1. The other components are considered as the environment in which the agent specification needs to be verified against. The details on the agent specification and environment specification will be discussed in the next section. Note that, monitor components have not received much attention in previous work because previous verification effort only focuses on a specific reasoning technique without considering them being embedded within a system interacting with the physical environment.

4 Verification framework

In this section we discuss the details of our proposed verification framework. Our framework employs the NuXmv model checker as the computation engine and uses the NuXmv specification language to specify the agent and the environment. We now describe the formalism of the verification framework.

4.1 Discrete decision making

We consider discrete decision making components implemented as policies, a popular representation of decision making logic as a result of advanced planning or learning processes [15, 6, 14]. Note that, our framework is not restricted to this specific decision making representation and can be extended to support different discrete decision making mechanisms such as agent programming languages. For instance, it has been shown in [7] that a complex set of robot decision making rules can be automatically translated to the NuXmv modeling language.

A decision making policy is formally defined as follows.

Definition 1. *A decision making policy is a tuple* $\sum = (\mathbf{S}, \mathbf{A}, \pi)$ *where*

- $\mathbf{S} = \{S_1, S_2, \ldots, S_n\}$ *is a set of* n *discrete state variables, where each state variable* S_i *takes on values in some finite domain* $Dom(S_i)$*. We call* $\mathbf{S}$ *the state vector.*
- $\mathbf{A} = \{a_1, a_2, \ldots, a_m\}$ *is a set of* m *actions.*
- $\pi : S_1 \times S_2 \times \cdots \times S_n \rightarrow \mathbf{A_{exec}}$ *is a complete function that maps each value of the state vector to a set of actions* $\mathbf{A_{exec}} \in \mathcal{P}(\mathbf{A})$*, where* $\mathcal{P}(\mathbf{A})$ *is the power set of* $\mathbf{A}$*.*

A decision making policy consumes the value of the state vector $\mathbf{S}$ provided by the monitors and decides a set of actions $\mathbf{A_{exec}}$ to be executed at each decision making cycle. The encoding of a decision making policy in a NuXmv specification is straightforward. Each state variable is represented by a NuXmv variable with the enumerated type. One remark is that a decision making policy is often very large with many state-vector-to-action-set mappings. While it is possible to represent each mapping in the NuXmv specification language, it would result in a very large specification which is computationally expensive to verify. Due to that, we first transform decision making policies to a compact format. Since a decision making policy can be seen as a multi-output truth table where the state vector values are inputs and the actions are Boolean outputs, one can apply two-level logic minimization [3] to reduce the truth table to a set of disjunctive normal forms (DNFs), that is, a disjunction of conjunctive clauses. Each DNF represents the condition on the state vector in which an action is executed at each decision making cycle.

We represent each action in the decision making policy as a symbol associated with its corresponding DNF. At each decision making cycle, if the symbol is *true*, that is, the DNF holds, the action is executed. Note that, the compact representation of decision making policies is not the main contribution of this paper. We aim at proposing a general framework in which different discrete decision making mechanisms can be specified and verified.

4.2 Monitor

Monitor components discretize information provided by perception algorithms. At each decision making cycle, each state variable in the state vector is updated by a monitor based on the most recent information received from the perception algorithms. The monitor of a state variable is formally defined as follows.

Definition 2. *A monitor M of a state variable is defined by a tuple $M = (I, S, s_{init}, \delta)$ where*

- *I is a set of inputs. Each input can be either a numerical value, an enumerated type value or an event.*
- *S is a finite, non-empty set of enumerated values. S is the domain of the monitored state variable.*
- *$s_{init} \in S$ is the initial value of the state variable.*
- *$\delta = I \times S \to S$ is the value transition function.*

A monitor is essentially a finite-state machine (FSM) where each state of the FSM corresponds to a value in the domain of the monitored state variable. The three types of monitor inputs cover most of possible information provided by perception algorithms. For example, spatial information such as locations, distances and maps can be represented by numerical values, object classification results can be represented by enumerated type values and notifications from other components or human operators can be represented by events.

Each monitor manipulates the value of the corresponding state variable based on the monitor's inputs. An event input can be represented as a Boolean variable that has the value *true* if the event is triggered. Enumerated type is supported natively in NuXmv and numerical type can be either real number, integer number or bounded integer number, depending on the concrete scenario or the fidelity level of the environment.

In our framework, the inputs of monitors represent the operating environments of agents. For example, a grid-map environment can be encoded using numerical inputs. A detail analysis on the scalability of the proposed framework, for example, how the size of the environment impacts the verification performance, is the subject of future work.

4.3 Environment assumptions

Instead of building a model for the environment, in the proposed verification framework, one can specify the environment as a set of LTL assumptions on the inputs of monitors and the values of action symbols which represent whether an action is executed at each decision making cycle.

An LTL formula specifies a condition on an infinite execution trace of the system. LTL extends first-order logic with time notion via supporting temporal operators such as *always* (G) and *eventually* (F). In this work we use an extended version of LTL supported by NuXmv that also consists of past operators such

as *previous state* (Y). For a complete syntax of the LTL version used, we refer readers to [2].

An LTL formula holds if it is true for every possible infinite execution trace of the system. An LTL property is verified against the conjunction of environment assumptions by checking whether the following LTL formula holds.

$$\phi_{assumptions} \Rightarrow \phi_{property} \tag{1}$$

4.4 Trade-off between completeness and fidelity

The underlying model checker used in our framework, NuXmv, supports the verification of both finite and infinite systems. The main trade-off between the completeness of the verification and the fidelity level of environment models is made on the specification of numerical variables in the set of monitor inputs I. If a numerical variable has an infinite domain, that is, its type is real or unbounded integer, the system becomes infinite. For infinite systems, NuXmv employs approaches extended from Bounded Model Checking (BMC) to perform verification. BMC verifies a system by searching for a counter-example of a bounded length that violates properties. BMC is incomplete as it might miss a counter-example if the bounded length is not large enough.

For the verification to be complete, the specified system must be finite. As numerical monitor inputs such as distance and battery level are often infinite, one needs to represent them approximately as bounded integers. Doing so makes the system finite at the cost of lowering the fidelity level of environment models. Depending on concrete applications, such approximation might affect the validity of verification results.

5 Example

We demonstrate our framework through an example where an autonomous UAV performs pylon inspection. The mission of the UAV is to autonomously visit predefined inspection points around pylons while taking into account safety requirements. The UAV system in the example was developed within the scope of the SafeDroneWare project[1] and has been deployed in a real UAV platform.

5.1 Monitor inputs

The following monitor inputs are provided by the perception components.

1. *distance_to_point*[3]: The flying distances between the current position of the UAV to each inspection point. In this model we assume that there are three inspection points.
2. *distance_to_landing_location*: The flying distance between the current position of the UAV to the landing location.

[1] https://www.imec-int.com/en/what-we-offer/research-portfolio/safedroneware

3. *altitude*: The current altitude of the UAV.
4. *battery*: The current battery level of the UAV.
5. *manual_control_on*: A request from the human operator to manually control the UAV.
6. *manual_control_off*: A request from the human operator to stop manually controlling the UAV.
7. *mission_start*: The human operator gives the UAV permission to start the mission.
8. *mission_abort*: A request from the human operator to abort the mission.
9. *configure_mission*: The mission has just been configured.
10. *communication_status*: The status of the communication link, which is continuously monitored by a third-party software component. The communication link can have three states: *stable*, *degraded* and *lost*.
11. *obstacle_detection_status*: The status of the obstacle detection component. A perception component is responsible for monitoring the state of all the hardware and algorithms performing obstacle detection. The state of the obstacle detection component can be either *stable* or *lost*.

The monitor inputs 1-4 are of numerical type, the inputs 5-9 are events and the remaining inputs are of enumerated type.

5.2 Monitors

The decision making policy of the UAV contains 16 state variables. The monitor of each state variable is specified using the NuXmv specification language according to Definition 2. Due to the lack of space, we only discuss three representative monitors. The complete specification is available online[2].

The state variable $S\_manual\_control\_request \in \{on, off\}$ represents whether the human operator wants to control the UAV manually. This state variable is updated based on two events *manual_control_on* and $manual\_control\_off$. As shown in Figure 2, every time the UAV receives a *manual_control_on* event, $S\_manual\_control\_request$ turns to *on* until a $manual\_control\_off$ event is received.

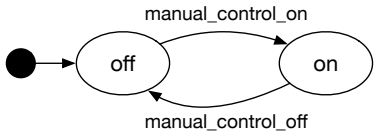

Fig. 2. The FSM for the state variable $S\_manual\_control\_request$.

The state variable $S\_battery \in \{ok, low, critical\}$ indicates the battery level of the UAV based on predefined thresholds. Figure 3 illustrates the FSM of the monitor for the $S\_battery$ variable.

[2] https://github.com/hoangtungdinh/paams20-supplemental-material

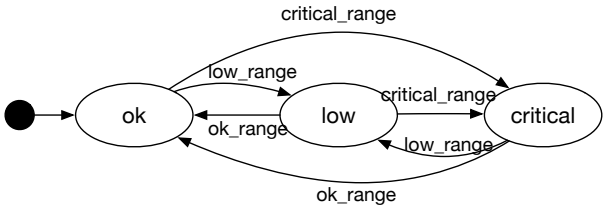

Fig. 3. The FSM for the state variable *S_battery*. The ranges are defined as follows.
critical_range : $battery < critical\_threshold$
low_range : $critical\_threshold \leq battery < low\_threshold$
ok_range : $battery \geq low\_threshold$

The state variable *S_pylon_inspection* keeps track of whether the UAV has inspected all the predefined points. To model the monitor of this state variable, we define an extra variable *point_index* to keep track of inspected points. Note that, the UAV must visit the points in a predefined order. Every time the current point is reached, that is, the distance to the point is 0, the value of *point_index* is increased. *S_pylon_inspection* turns to *complete* when all the points are visited. Figure 4 shows the FSMs of *point_index* and *S_pylon_inspection* with three predefined inspection points.

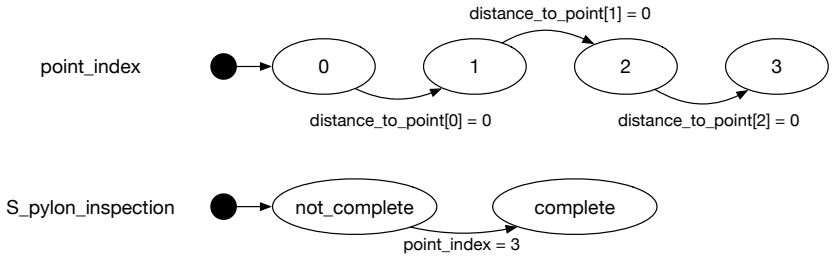

Fig. 4. The FSMs representing *point_index* and *S_pylon_inspection*.

5.3 Properties and environment assumptions

We present three example properties to be verified.

1. The UAV must always land at the predefined landing location.

$$G(altitude = 0 \Rightarrow distance\_to\_landing\_location = 0) \tag{2}$$

2. The UAV must only fly once it has received the `mission_start` event. (The LTL operator O means that the expression holds in at least one previous time step).

$$G(altitude > 0 \Rightarrow O(mission\_start = true)) \tag{3}$$

3. The UAV must eventually visit all the predefined points.

$$F(point\_index = 3) \tag{4}$$

During the verification process, we derive environment assumptions, both by our application knowledge and by counter-examples returned by NuXmv.

The first type of assumptions is the effects of the action execution on the environment. For example, the *altitude* of the UAV will increase when the UAV executes the $take\_off$ action and will decrease to 0 only when the UAV executes the land action.

$$G(take\_off \Rightarrow next(altitude) > altitude) \tag{5}$$

$$G((next(altitude) = 0 \wedge altitude > 0) \Rightarrow land) \tag{6}$$

Another assumption is that the UAV only changes its location, that is, the distances between the UAV and other locations are changed, by executing actions involving the movement of the UAV. Also, it can only change its location while it is flying.

$$\begin{aligned} &G((next(distance\_to\_point) \neq distance\_to\_point \vee \\ &next(distance\_to\_landing\_location) \neq distance\_to\_landing\_location) \\ &\Rightarrow ((go\_to\_landing\_location \vee manual\_control \vee go\_to\_point) \wedge \\ &altitude > 0)) \end{aligned} \tag{7}$$

We also need to assume that at the beginning, the UAV is landed at the landing location.

$$altitude = 0 \wedge distance\_to\_landing\_location = 0 \tag{8}$$

Note that, not all properties require the same assumptions. For example, the following assumptions are specifically to guarantee Property 3. For the UAV to complete the inspection, it must eventually receive a $configure\_mission$ event and a *mission_start* event. In addition, the human operator must never abort the mission, that is, the UAV must never receive a *mission_abort* event.

$$F(configure\_mission = true) \tag{9}$$

$$F(mission\_start = true) \tag{10}$$

$$G\neg(mission\_abort = true) \tag{11}$$

The last example assumption is that eventually, the UAV is never blocked.

$$\begin{aligned} &FG(go\_to\_point \wedge point\_index < 3 \wedge \\ &next(distance\_to\_point[point\_index]) < distance\_to\_point[point\_index]) \end{aligned} \tag{12}$$

Due to the lack of space, we do not include all assumptions and properties. However, we believe that the example assumptions and properties are representative enough to understand the proposed framework. We verified the properties

with two different models. In the first model, all numerical monitor inputs are specified as of type real. In the second model, all numerical monitor inputs are specified as of type bounded integer so that NuXmv can apply complete model checking techniques. In both models, NuXmv is able to verify all properties and return counter-examples if a property is violated within a few seconds. The generated counter-examples were useful as they helped us realize possible failure situations that we did not think of at the design time. Thanks to the counter-examples, we were able to derive and state explicitly all necessary assumptions for the properties to hold.

6 Conclusions

Verifying autonomous robotic agents is challenging due to the complexity of their operating environments. In this paper, we propose a framework to model and verify the decision making of autonomous robotic agents. The proposed framework makes a clear separation between the agent modeling and the environment modeling. Formalism to specify the discretization logics of the perception information is provided. The environment in the proposed framework is specified as a set of LTL assumptions on perception information. An example of verifying an autonomous UAV performing pylon inspection was provided to demonstrate the usability of the framework.

We experience that many complicated assumptions can be represented in our framework, although it requires expert knowledge in LTL to encode the assumptions. Future studies are required to validate and explore the applicability of the framework on different robotic agent systems, as well as to analyze the computational complexity of the framework. Moreover, patterns for specifying assumptions on different parts of systems such as actions' effects can be derived. We also plan to extend the framework so that it can take into account the verification of motion planning and control components.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven. We thank the anonymous reviewers for their helpful comments.

References

1. Aminof, B., Murano, A., Rubin, S., Zuleger, F.: Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. pp. 1190–1199. AAMAS '16, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2016)
2. Bozzano, M., Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: nuXmv 1.1. 1 User Manual (2016)

3. Coudert, O., Sasao, T.: Two-level logic minimization. In: Logic Synthesis and Verification, pp. 1–27. Springer (2002)
4. Dennis, L.A., Fisher, M., Lincoln, N.K., Lisitsa, A., Veres, S.M.: Practical verification of decision-making in agent-based autonomous systems. Automated Software Engineering **23**, 305–359 (Sep 2016)
5. Dixon, C., Webster, M., Saunders, J., Fisher, M., Dautenhahn, K.: "The Fridge Door is Open"–Temporal Verification of a Robotic Assistant's Behaviours. In: Mistry, M., Leonardis, A., Melhuish, C. (eds.) Advances in Autonomous Robotics Systems, vol. 8717, pp. 97–108. Springer International Publishing, Cham (2014)
6. Fu, J., Ng, V., Bastani, F., Yen, I.L.: Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
7. Gainer, P., Dixon, C., Dautenhahn, K., Fisher, M., Hustadt, U., Saunders, J., Webster, M.: CRutoN: Automatic Verification of a Robotic Assistant's Behaviours. In: Critical Systems: Formal Methods and Automated Verification, pp. 119–133. Springer (2017)
8. Ingrand, F., Ghallab, M.: Deliberation for autonomous robots: A survey. Artificial Intelligence **247**, 10–44 (2017)
9. Kouvaros, P., Lomuscio, A., Pirovano, E., Punchihewa, H.: Formal Verification of Open Multi-Agent Systems. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. pp. 179–187. AAMAS '19, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2019)
10. Lomuscio, A., Pirovano, E.: A Counter Abstraction Technique for the Verification of Probabilistic Swarm Systems. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. pp. 161–169. AAMAS '19, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2019)
11. Luckcuck, M., Farrell, M., Dennis, L.A., Dixon, C., Fisher, M.: Formal Specification and Verification of Autonomous Robotic Systems: A Survey. ACM Comput. Surv. **52**, 100:1–100:41 (Sep 2019)
12. Morse, J., Araiza-Illan, D., Lawry, J., Richards, A., Eder, K.: Formal Specification and Analysis of Autonomous Systems under Partial Compliance. arXiv:1603.01082 [cs] (Mar 2016)
13. Rubin, S.: Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 199–208. AAMAS '15, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2015)
14. Shalev-Shwartz, S., Shammah, S., Shashua, A.: Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. arXiv:1610.03295 [cs, stat] (Oct 2016)
15. Spaan, M.T.J., Veiga, T.S., Lima, P.U.: Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. Autonomous Agents and Multi-Agent Systems **29**, 1157–1185 (Nov 2015)
16. Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K.L., Dautenhahn, K., Saez-Pons, J.: Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study. IEEE Transactions on Human-Machine Systems **46**, 186–196 (Apr 2016)
17. Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K.L., Dautenhahn, K.: Formal verification of an autonomous personal robotic assistant. Proc. AAAI FVHMS pp. 74–79 (2014)