Assuring Learning-Enabled Increasingly Autonomous Systems*

Nandith Narayan
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne FL, USA
nnarayan2018@my.fit.edu

Parth Ganeriwala
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne FL, USA
pganeriwala2022@my.fit.edu

Randolph M. Jones
Applied Cognitive Systems
Soar Technology
Ann Arbor MI, USA
rjones@soartech.com

Michael Matessa
Applied Research and Technology
Collins Aerospace
Cedar Rapids IA, USA
mike.matessa@collins.com

Siddhartha Bhattacharyya
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne FL, USA
sbhattacharyya@fit.edu

Jennifer Davis
Applied Research and Technology
Collins Aerospace
Cedar Rapids, IA U.S.A.
jen.davis@collins.com

Hemant Purohit
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne FL, USA
hpurohit2021@my.fit.edu

Simone Fulvio Rollini
Applied Research and Technology
Collins Aerospace
Rome, Italy
simonefulvio.rollini@collins.com

2023 IEEE International Systems Conference (SysCon) | 978-1-6654-3994-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/SysCon53073.2023.10131227

***Abstract*—Autonomous agents are expected to intelligently handle emerging situations with appropriate interaction with humans, while executing the operations. This is possible today with the integration of advanced technologies, such as machine learning, but these complex algorithms pose a challenge to verification and thus the eventual certification of the autonomous agent. In the discussed approach, we illustrate how safety properties for a learning-enabled increasingly autonomous agent can be formally verified early in the design phase. We demonstrate this methodology by designing a learning-enabled increasingly autonomous agent in a cognitive architecture, Soar. The agent includes symbolic decision logic with numeric decision preferences that are tuned by reinforcement learning to produce post-learning decision knowledge. The agent is then automatically translated into nuXmv, and properties are verified over the agent.**

***Index Terms*—Formal Verification, Human-Autonomy Interactions, Human-Autonomy Interface, Assuring Learning Enabled Systems, Autonomous Agent Verification**

I. Introduction

With the advancements in machine learning, it is expected that autonomous agents/systems will become increasingly intelligent. As a result, it is envisioned that autonomous agents will be capable enough to handle emerging situations. This is evident from the research explorations that are evaluating advanced capabilities such as autonomous collision avoidance, advanced air mobility, drone delivery systems, and autonomous cars. Newton et. al [1] discuss the need for the integration of new advanced algorithms to ensure safety of passengers and bystanders for autonomous Urban Air Mobility (UAM). They also discuss how to assess the vehicle capability to handle contingency scenarios. Gregory et. al [2] discuss the need for the use of deterministic and learning models to handle uncertain situations with the design of intelligent systems and the assessment of such vehicles. Although these advancements enhance the expertise of the autonomous system/agent, it makes assurance and certification more challenging.

In the research effort by Bhattacharyya et. al [3] [4], they discuss the design of a framework to include Human-Machine Interfaces and formal methods to verify the correctness of an Increasingly Autonomous System (IAS). In that effort the IAS does not include machine learning. For the assurance of learning-enabled IAS designed for contingency management, Neogi et. al [5] investigate chunking, a general learning mechanism in Soar [6]. They do not evaluate the need for human-machine interactions, and the learning mechanism is not based on machine learning using statistical inferences.

In the Assured Human Machine Interface for Increasingly Autonomous Systems (AHMIIAS) approach [3], along with the integration of advanced technologies, we have identified that there is a need to discover the human-machine interaction that should be implemented for learning-enabled systems. In section II related work is discussed, with the proposed methodology in section III. In section IV, the human-machine interactions essential for our selected scenarios and the learning function of the IAS are identified and discussed. The

*We would like to thank Anubhav Gupta for his initial support in setting up the infrastructure for LEIAS. We would also like to thank Natasha Neogi and Paul Miner of NASA LaRC for their feedback on the learning function and its safety verification. This work was funded by award 80NSSC20M0005 to the NASA Langley Research Center from the NASA Shared Services Center.

design of, and the experiments performed with, the learning-enabled IAS are explained in section V. Then, in section VI, formal verification of the IAS is elaborated on, with the verification of learning-enabled autonomous systems. Finally, the results are in section VII with conclusion in section VIII.

Our contribution is the design of a framework that integrates human-machine interactions and formal methods for the assurance of learning-enabled autonomous systems. We have identified the human-machine interactions that are essential for humans to team up with autonomous systems that can perform machine learning. We have extended our Soar-to-nuXmv translator to support constructs needed for learning in Soar. We have verified correctness and safety properties over the translated learning-enabled agent. The architecture models, IAS agent, translator code, and verified models can all be found on our project repository[1].

II. Related work

In the pursuit of increasing the learning capability of IAS, various methodologies, including deep learning, Markov decision processes, data analysis, and real-time discoveries have been proposed to achieve significant enhancements in cognitive autonomy [7]. On the other hand, within the controls research community, constrained control approaches such as Model Predictive Control (MPC) ensure the learning based on the previously recorded data. MPC has gained the researchers' attention on the basis of the methods being used in learning that include learning through system dynamics, learning the controller design, and model predictive control for safe learning [8]. Learning-Enabled Systems (LESs) have also been designed on the basis of goal-based, model-driven approaches to assure self-adaptation and self-assessment [9]. Additionally, various uncertainties about the safety assurance of LESs have been exhibited in Model-Driven Assurance for Learning-enabled Autonomous Systems (MoDALAS) [9].

Formal methods have been consistently used in assuring safety and trust in safety critical applications, intelligent systems, and co-operative autonomous agents [4]. Cofer et al. [10] discuss a formalized run-time assurance architecture together with diverse monitors that were used to ensure the safety of an aircraft taxiing application that included a neural-network-based learning-enabled component. Moreover, the research conducted by Bhattacharyya et. al [3] shows that the human-machine team architecture can be designed and verified using Architecture Analysis and Design Language (AADL) [11] and the Assume Guarantee REasoning Environment (AGREE) [12], which lays a strong foundation in increasing trust and safety. This paper extends [3] by the addition of learning to the IAS to create a Learning-Enabled IAS (LEIAS), updating the architecture models, extending the Soar-to-nuXmv translator to support learning, and verifying properties related to outcomes of learning.

[1]https://github.com/loonwerks/AHMIIAS/tree/Year_2

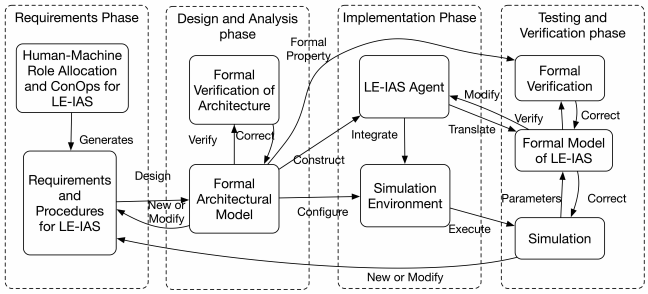

Fig. 1. Proposed methodology for LEIAS

III. Research Methodology

The proposed methodology for the design of LEIAS is shown in Figure 1. During the Requirements phase, the scenarios for learning are developed, along with the specifications for the human-machine interfaces. These are then mapped into the architectural components that are required for learning to occur. Then, the behavior of the components are added in the Implementation phase, as learning infrastructure, the design of the learning agent and, then, experiments are performed for the LEIAS to learn the specified behavior. Finally, during the Testing and Verification phase, formal verification of the LEIAS is performed by extending the previous automated translator with the inclusion of the translation of learning-related constructs.

IV. HMI and Architecture of LEIAS

A. Human-Machine Interface (HMI)

Bhattacharyya et. al [3] developed two scenarios to test the collaborative roles and responsibilities of a human pilot and an IAS in Urban Air Mobility (UAM) operations. The first scenario, the Unreliable Sensor Scenario, introduces an off-nominal situation where the GPS sensor's reliability is reduced due to an urban canyon. The IAS detects the unreliable GPS sensor and determines the correct position using Lidar and IMU, informing the pilot about the situation. The pilot can either accept or reject the IAS interpretation of the situation. To extend the Unreliable Sensor Scenario with learning, the IAS was trained to learn the pilot preferences. As the learning occurred, the specifications for the HMI were established to explain the pilot what the IAS learnt. With the LEIAS methodology, the IAS learns about the pilot's preferences and informs the pilot of an unreliable sensor accordingly. However, as soon as the error value passes the safety threshold, the LEIAS will always inform the pilot of the unreliable sensor. In the second scenario, the Aborted Landing Scenario, the pilot detects an unsuitable landing area, aborts the landing, and relies on the IAS to calculate alternate landing options. The IAS presents the best option and reasoning to the pilot, who can either accept or choose an alternate landing site. In both scenarios, the IAS supports the pilot by monitoring the situation, assisting in decision making, and providing information, while the pilot retains the ultimate decision-making authority.

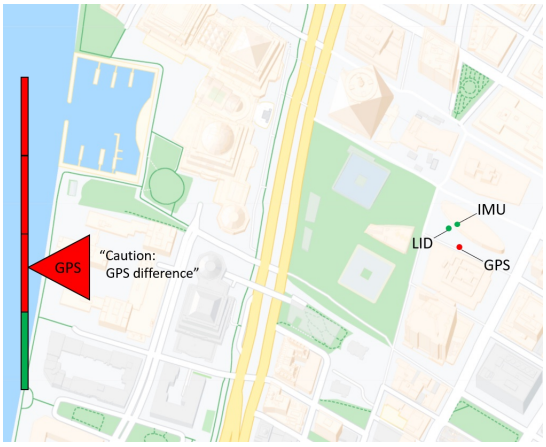

Fig. 2. Learning HMI

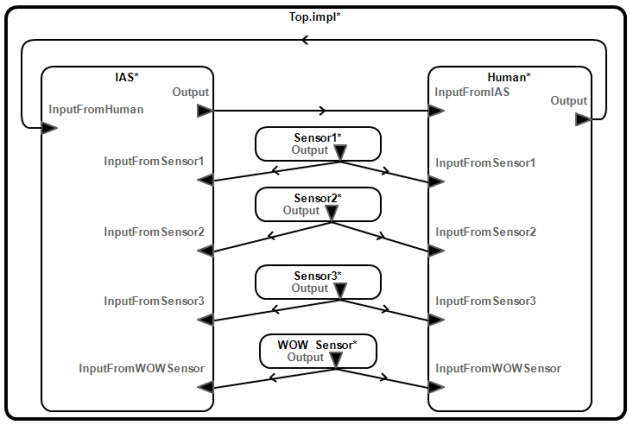

Fig. 3. Human-IAS team model

The HMI display consists of a map, position indicators (for GPS, Lidar, and IMU sensors), and a triangular unreliable sensor alerting icon placed next to a severity indicator bar. The position indicators show the pilot any discrepancy between sensor positions in units that a pilot can understand (e.g., within a building, within a block, or across town).

The error range is decomposed into four segments: a normal range for minimal sensor discrepancy, a level 1 range for larger discrepancy, a level 2 range for even larger discrepancy, and a safety range for a significant sensor discrepancy. The IAS always provides an alert for the safety range, never provides an alert for the normal range, and learns the pilot's preference for level 1 and level 2; it colors the corresponding bar segment red if an alert is to be given and green if an alert is not to be given.

The main objective of these scenarios are to capture the collaboration between the IAS and the pilot. In the Unreliable Sensor Scenario, the IAS detects that the GPS sensor is unreliable by comparing the GPS position value with the values from the Lidar and IMU sensors. The errors are abstract values that are induced to indicate the pairwise difference in position. It does not include the inner workings of the sensor. The scenario assumes that in this specific situation, the Lidar and IMU sensors are more reliable than the GPS sensor. However, it should be noted that Lidar and IMU sensors can also experience errors, which was also evaluated. The scenarios can be modified to explore different failure modes or sensor combinations.

B. Human-Machine Team Architecture Model

We capture the structure of the model (components and interfaces) in AADL. AADL is a standardized language for embedded, real-time systems. It supports design, analysis, virtual integration, and code generation. The graphical representation of the AADL model, showing components and connections, is given in Fig. 3. We include in the model the human pilot, the IAS, a Weight on Wheels (WOW) sensor/subsystem, and three position sensors. We capture an abstract representation of the behavior of the system and each subcomponent in an assume-guarantee contract in the AGREE. AGREE performs compositional analysis, verifying system requirements based on the composition of the component assume-guarantee contracts. By abstracting the implementation of subsystems and software components into formal contracts, large systems can be verified hierarchically without the need to perform a monolithic analysis of the entire system at once. AGREE translates the model to the Lustre language and then performs verification using a model checker (e.g., JKind [13]) and an SMT solver (e.g., Z3 [14]).

The structure of the human-machine team architecture model builds on the AHMIIAS framework introduced by Bhattacharyya et. al [3] in the following way. First, the IAS interface is extended to allow the communication of the sensor discrepancy value, the corresponding range, and the presence of an alert, for each of the three sensors; the pilot's interface is extended to provide an acceptance/rejection feedback to IAS alerts. Second, requirements are added in terms of guarantees in the AGREE language, to specify the IAS behavior and to express expectations on the pilot's behavior. For example, the IAS (i) should not issue an alert for a sensor if the sensor discrepancy falls into the normal range; and, on the contrary, (ii) should issue an alert for a sensor if the sensor discrepancy falls into the safety range:

```
guarantee ias_g1 "If the sensor error is in Normal range, then no alert is issued":
    (Output.sensor1ErrorData.errorRange = enum(Types_Constants::SensorErrorRange, Normal)
        => (not Output.sensor1ErrorData.errorAlert))
and (Output.sensor2ErrorData.errorRange = enum(Types_Constants::SensorErrorRange, Normal)
        => (not Output.sensor2ErrorData.errorAlert))
and (Output.sensor3ErrorData.errorRange = enum(Types_Constants::SensorErrorRange, Normal)
        => (not Output.sensor3ErrorData.errorAlert));

guarantee ias_g2 "If the sensor error is in Safety range, then an alert is issued":
    (Output.sensor1ErrorData.errorRange = enum(Types_Constants::SensorErrorRange, Safety)
        => Output.sensor1ErrorData.errorAlert)
and (Output.sensor2ErrorData.errorRange = enum(Types_Constants::SensorErrorRange, Safety)
        => Output.sensor2ErrorData.errorAlert)
and (Output.sensor3ErrorData.errorRange = enum(Types_Constants::SensorErrorRange, Safety)
        => Output.sensor3ErrorData.errorAlert);
```

Fig. 4. IAS alert issuing

Third, the requirements on the system, i.e., on the interaction between the IAS and pilot, are encoded as AGREE lemmas and subject to formal verification.

Whenever a sensor discrepancy is in the level 1 or level 2 range, the learning process comes into play. Broadly speaking, after the pilot's response to an alert in the presence of a certain sensor discrepancy, in case of acceptance (resp. rejection), it should become more (resp. less) "likely" for the IAS to issue an alert for that sensor and discrepancy. The definition of the notion of likelihood, as well as of the details of how it is affected by the pilot's response, depend on the particular learning algorithm and implementation; for these reasons, they are not represented at the architecture level. However, the IAS-pilot communication flow, that also enables learning, is modeled by means of guarantees both on the IAS and the pilot's side; for example, the pilot is expected to provide feedback to an alert from the IAS in a reasonable amount of time (only guarantees related to one of the sensors are reported for brevity):

```
guarantee h_g1 "If a sensor 1 alert is received by the operator,
            then an alert response is communicated by the operator at the next step":
    (prev(InputFromIAS.sensor1ErrorData.errorAlert, false))
    <=> (Output.sensor1AlertResponse = enum(Types_Constants::SensorAlertResponse, Accept) or
         Output.sensor1AlertResponse = enum(Types_Constants::SensorAlertResponse, Reject));

guarantee h_g2 "If no sensor 1 alert is received by the operator,
            then no alert response is communicated by the operator at the next step":
(prev(not InputFromIAS.sensor1ErrorData.errorAlert, true))
    <=> (Output.sensor1AlertResponse = enum(Types_Constants::SensorAlertResponse, None));
```

Fig. 5. Pilot's response to alert

V. LEIAS

The design of a learning-enabled IAS in this research effort involves using and evaluating learning capabilities in the cognitive architecture Soar.

A. Learning in the Soar Cognitive Architecture

The problem of V&V for intelligent systems is further complicated when those systems are able to learn and improve their behavior with experience. As our case study focuses in part on an IAS implemented in the Soar cognitive architecture [6], here we describe reinforcement learning, as that is what was integrated for this study. Soar supports other forms of learning, i.e. chunking, episodic learning and semantic learning [6].

Reinforcement learning is a well established form of learning in the machine learning research community [15]. Soar's particular implementation of reinforcement learning integrates the learning algorithms with Soar's native decision-making structure [16]. In Soar, decisions are made by observing the current state of the world, suggesting candidate inferences or actions ("operators") to execute next, determining which of those candidates should be most preferred in the current context, and then selecting a single inference or action to execute. This decision cycle repeats quickly and indefinitely to implement a Soar agent's intelligent reasoning and behavior.

B. Design of the Integrated Learning Agent

Our research effort initially focused on developing a Soar decision-making agent with no learning [3]. This agent makes a relatively simple set of decisions about how to control the aircraft during different phases of flight, including takeoff, transit, and landing, as well as transitions between these phases. The agent also includes some knowledge about how to address situations in which it infers that one of its position sensors is faulty. This includes decision-making logic about alerting a human pilot and using the pilot's guidance about whether to decide that a sensor is unreliable.

The use case focuses on the design of an intelligent IAS that learns the human pilot's preferences about the conditions in which the pilot wants to be alerted about a potential sensor failure.

Symbolic Decision Logic-Based Learning Agent. The LEIAS agent is structured based on regions of alerting levels as described in Section IV B. For normal range of error, the system will never alert the pilot. For safety range of error, the system will always alert the pilot. For errors in level 1 or 2, initially, the Soar agent decides to generate an alert. Over time, the system is given a reward when it alerts the pilot and the pilot "accepts" the alert, and a punishment when the pilot "rejects" the alert. With each new experience, the reinforcement learning algorithm incrementally allocates portions of these reward/punishment signals to the decisions.

C. Experiments for LEIAS

To conduct the experiments, the overall learning architecture included the integration of the learning infrastructure, the interactions with the environment, and the symbolic decision logic-based learning agent. Cycles of training and testing were executed to validate that learning.

The **learning infrastructure** consisted of all the necessary entities and interactions that support the generation of data. One of the infrastructural requirements was the need to store the data, as reinforcement learning requires several cycles to process it. The second requirement was to store the threshold values and the third requirement was to store the pilot's response. To expedite data collection, the pilot's response was scripted as part of the infrastructure.

D. Experimental Setup and Results for LEIAS

The learning process was split into a series of learning trials and testing trials. During each learning trial, a random error is introduced and the IAS decides whether to warn the pilot. If the IAS decision matches the preferences of the scripted pilot, then the agent is rewarded with a value of +1. If the IAS decision does not match the scripted pilot, then the agent is given a reward value of -1. In our scenario, the scripted pilot wants the IAS agent to always warn when the error is greater than or equal to 9 (level 2). During each testing trial, the introduced sensor error continuously increases over time until it reaches safety error range. These trials validated the agent's learned behavior. Boltzmann with high and low temperatures, and simulated annealing were explored as the rule selection policy. Simulated annealing utilize gradual learning which involves starting with a high learning rate or temperature and gradually decreasing it over time. A high temperature allows the algorithm to explore a wider range of solutions, while in Boltzmann learning, a high acceptance rate enables the algorithm to accept worse moves. As the temperature

or acceptance rate decreases, the algorithm focuses more on refining the current solution. However, selecting an appropriate cooling schedule and temperature is crucial for stability and responsiveness. If the cooling is too fast or too slow, the algorithm may get stuck in a sub-optimal solution or take too long to converge to an optimal solution.

Results. Fig. 6 shows the outcomes of the experiments applying Boltzmann policy with high temperature of 25. The bar graph in Fig. 6 at the bottom displays the pairwise error values of the inputs received. The graph in the top left corner indicates that the LEIAS gets rewarded to warn every time the error is in level 2; the graph in the top right corner shows that the LEIAS gets negatively rewarded every time it warns when the error value is in level 1 (between 8 and 9), so it learns not to warn when the error is below 9. Similar behaviors were observed in Fig. 7 (Boltzmann with low temperature) and Fig. 8 (simulated annealing). Boltzmann with high temperature explores the search space more than the other algorithms; as expected, this resulted in more gradual learning.

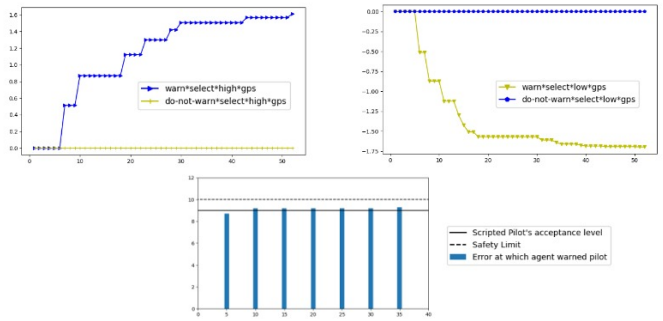

Fig. 6. Results with Boltzmann high temperature

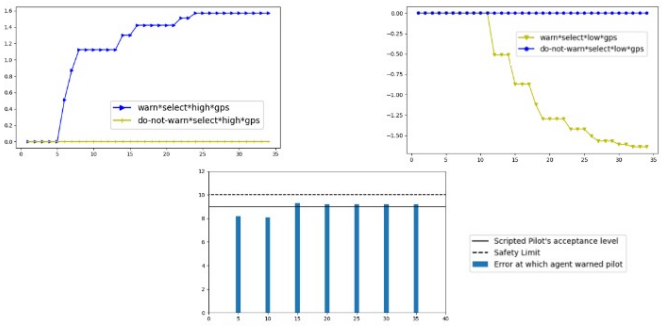

Fig. 7. Results with Boltzmann low temperature

VI. Verification of LEIAS

The verification process involved translation from Soar to nuXmv and then performing verification by means of the nuXmv tool. This includes sophisticated SMT-based model checking techniques and expands the NuSMV language with additional data types, such as integers and reals, for infinite-state systems [17].

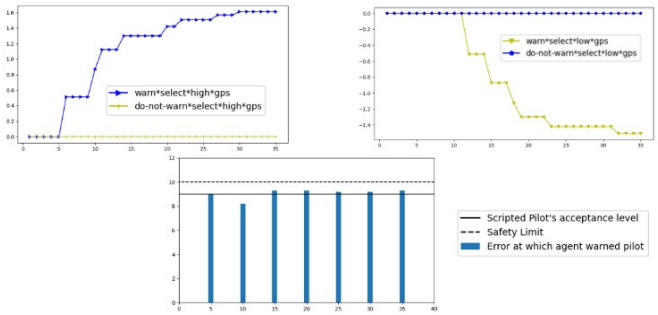

Fig. 8. Results with Simulated Annealing

A. Translation from Soar to nuXmv

The translation from the LEIAS built in Soar to nuXmv involves four phases: Input, Parsing, Refinement and Output (Fig. 9). During the Input phase, the Soar agent source file is transformed into a runtime execution file, by the Soar debugger. During the Parsing phase, the ANTLR [18] grammar representing the Soar grammar rules is used to create an abstract syntax tree, which is in turn used to extract relevant information. During the Refinement phase, the variable types and values are identified, followed by combining the pre and post conditions associated with each of the Soar rules. Finally, during the output phase, a nuXmv file is generated that consists of the variables, modules, transition and guard conditions, as is expected for a state machine representation of the LEIAS translated from Soar.

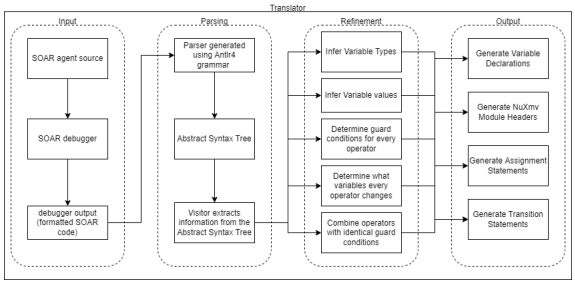

Fig. 9. Translation from LEIAS in Soar to nuXmv

To handle constructs associated with learning in Soar, the changes enumerated below were made to the translator:

- **Priority with preference values**: When using reinforcement learning, Soar prioritizes rules by assigning preference values, however, in nuXmv, priority is executed in a top-down order. The preferred order is computed and then generated by the translator. For example, *warn*select*high*gps* is a Soar production that stores the preference value to warn for GPS level 2 error. The production stores it as $< s > operator < o >= 0.0$, where $< s >$ represents the state which assigns a value of 0.0 to the operator $< o >$. As the agent learns about the pilot's preferences, this value is updated. During translation, the preference values are used to generate the rules from highest to lowest priority in nuXmv.

- **Elaboration rules**: The elaboration phase is when rules, that elaborate the scenario, propose operators, and assess operators are fired. The numeric mapping of symbolic constants used for learning a range of values in Soar was performed with the use of elaboration rules. The translation handled this mapping of symbolic constants to numeric values for the LEIAS.
- **Existence checks**: In Soar, the existence of a variable is checked, before the evaluation of any condition associated with the variable. As part of the learning process, some values needed to be removed before new values were assigned. As a result, the existence check was also translated for integers and real values. For example, some of the variables in nuXmv executing existence are *state_gps-error-info_current-value_exists = yes, state_safety-error_exists = no*.
- **Handling of impasse rules**: During the *propose* cycle, Soar identifies the set of rules for which the conditions match, then, during the *apply* cycle, it checks the preference values of the rules and selects the one that has the highest preference value. During the translation to nuXmv, the proposed rules were combined together; without the merging of these rules, nuXmv would have always selected the first apply operator that was generated. For example, the following nuXmv translation, *(state = run & state_ready-to-decide = yes): combined-warn-propose-do-not-warn-propose; translated the condition of whether the agent has set the ready-to-decide flag to yes and then sets the state_operator_name to combined-warn-propose-do-not-warn-propose which handles the impasse of proposing warn or do-not-warn operators.* Then the translator generated the apply rules based on preference values in a top-down order from highest to lowest preference value, as nuXmv selects the one that appeared first when the condition of more than one rule matches.

B. LEIAS Properties and Verification in nuXmv

The formal verification of the learning-enabled agent was performed by decomposing the rules of the LEIAS into two categories, one that was generated based on learning, and the other that focused on the non-learning aspects.

The LEIAS agent was trained to learn the threshold of the pairwise sensor (GPS, IMU, Lidar) error difference at which the pilot is warned, or not warned. The preference values were recorded as learning occurred. Then the rules were translated into nuXmv. An input template was created in nuXmv for the generation of the input values that the IAS agent responds to. These input values updated the learning parameters, along with the change of operational parameters. The formal representations of the linear temporal logic specifications (LTLSPECs) in nuXmv were derived by decomposition from the learning specifications in English, which were at a higher level of abstraction. 10 properties were verified with a total of 54 LTLSPECs for the three sensors being used for the entire verification process. For each of the properties, counter-properties were also formulated to verify the completeness and record any anomalies or edge cases for safety assessment.

TABLE I
SPECIFICATIONS FOR LEIAS BEHAVIOR

Learning Specification in English	**Formal Specification**
If the sensor error is at or past the safety threshold, the agent shall issue a warning.	LTLSPEC F (X state_imu-error-info_warn-condition = safety & state_operator_name = warn)
If error difference within sensors is non zero, then the agent shall execute the error changed operator	LTLSPEC F (X state_imu-error-info_current-value != state_imu-error-info_old-value & state_operator_name = record-changed-error-imu)
If the difference in error values hasn't changed, the agent shall not execute the error changed operator	LTLSPEC G (X state_operator_name = record-changed-error-imu → state_imu-error-info_value-changed = nil & (state_imu-error-info_current-value = state_imu-error-info_old-value))
When a sensor is at fault, and the warning level is low, the agent shall not warn	LTLSPEC F (X state_imu-error-info_warn-condition = low & state_operator_name = warn)
When a sensor is at fault, and the warning level is high, the agent shall warn	LTLSPEC F (X state_imu-error-info_warn-condition = high & state_operator_name = warn)
If the sensor error is within normal threshold, the agent shall not issue a warning	LTLSPEC F (X state_imu-error-info_warn-condition = normal & state_operator_name = do-not-warn)
A learned rule with the lowest preference shall never fire.	LTLSPEC F (X (state_imu-error-info_error_warn-condition = low) → state_operator_name = warn)
If a warning has previously been issued, the agent shall not issue another warning	LTLSPEC F (X (state_warning-issued != nil & state_imu-error-info_warn-condition = safety) → (state_operator_name = do-not-warn))
The agent shall resolve an impasse in selecting learned rules	LTLSPEC F (X (state_imu-error-info_warn-condition = high) → state_operator_name = combined-warn-propose-do-not-warn-propose))
The agent shall always follow the preference order from the learned rules	LTLSPEC F (X state_imu-error-info_warn-condition = normal → state_operator_name = do-not-warn)

The learning performed by the LEIAS was verified by first generating the verification queries written in English text shown in Table I.

VII. RESULTS AND DISCUSSION

The LEIAS was formally verified within the AHMIIAS framework, after an iterative interplay between formal verification and simulation, which identified errors in the IAS model, as well as errors in translation. In the course of experimentation and translator updates to better align the formal verification with the learning specifications, there were challenges that needed to be addressed due to limitations of nuXmv. For example, nuXmv constructs cannot handle tree-like structures and one of the instances of our implementation only had a single variable for current-value and old-value for the sensor errors associated with three types of errors. In Soar, due to the inherent tree structure, it corresponds to three current values, one for each of the errors, therefore retaining

that information. This was solved by flattening the hierarchical data structure present in Soar into three separate error variables *(state_gps-error-info, state_imu-error-info, state_lidar-error-info)* to match nuXmv constructs. Due to the abovementioned change, each rule corresponding to the sensor error had to be triplicated. To make it modular and simpler, from a software engineering standpoint, macros were introduced. Macros efficiently compressed the Soar code by reducing the need for 33 rules to 20 rules. The execution of macros required the JSOAR debugger output before translation in nuXmv, as the macros were fully realized through the debugger.

VIII. Conclusion and Future work

In this research effort, we demonstrated the integration of learning in the AHMIIAS framework. This was accomplished by identifying requirements for the Human Machine Interface for a learning system and the architecture for the human-machine team. Finally, the simulation and verification of the learning-enabled IAS was performed to check the correctness of the IAS. Our approach showed how LEIAS can be designed with cognitive modeling, translated to a formal environment, and validated and verified through simulation and formal verification. Future work will involve investigating the life cycle management of LEIAS. Additionally, runtime assurance methods will be evaluated to perform assurance checks during learning by the LEIAS.

References

[1] N. H. Campbell, M. J. Acheson, and I. M. Gregory, *Dynamic Vehicle Assessment for Intelligent Contingency Management of Urban Air Mobility Vehicles*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1001

[2] I. M. Gregory and et al., "Intelligent contingency management for urban air mobility." in *Dynamic Data Driven Applications Systems. DDDAS 2020. Lecture Notes in Computer Science*, 2020.

[3] S. Bhattacharyya, J. Davis, A. Gupta, N. Narayan, and M. Matessa, "Assuring increasingly autonomous systems in human-machine teams: An urban air mobility case study," *Electronic Proceedings in Theoretical Computer Science*, vol. 348, pp. 150–166, oct 2021. [Online]. Available: https://doi.org/10.4204%2Feptcs.348.11

[4] S. Bhattacharyya, N. Neogi, T. Eskridge, M. Carvalho, and M. Stafford, "Formal assurance for cooperative intelligent agents," in *NASA Formal Methods Symposium LNCS*, vol. 10811, 2018.

[5] N. Neogi, S. Bhattacharyya, D. Griessler, H. Kiran, and M. Carvalho, "Assuring intelligent systems: Contingency management for uas," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 9, pp. 6028–6038, 2021.

[6] J. Laird, *The SOAR Cognitive Architecture*. MIT Press, 2012.

[7] G. Mani, B. Bhargava, P. Angin, M. Villarreal-Vasquez, D. Ulybyshev, and J. Kobes, "Machine learning models to enhance the science of cognitive autonomy," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2018, pp. 46–53.

[8] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 269–296, 2020. [Online]. Available: https://doi.org/10.1146/annurev-control-090419-075625

[9] M. A. Langford, K. H. Chan, J. E. Fleck, P. K. McKinley, and B. H. Cheng, "Modalas: Model-driven assurance for learning-enabled autonomous systems," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2021, pp. 182–193.

[10] D. Cofer and et. al, "Run-time assurance for learning-enabled systems," in *NASA Formal Methods: 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 361–368. [Online]. Available: https://doi.org/10.1007/978-3-030-55754-6_21

[11] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, 1st ed. Addison-Wesley Professional, 2012.

[12] M. W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. P. E. Heimdahl, and S. Rayadurgam, "Your "what" is my "how": Iteration and hierarchy in system design," *IEEE Software*, vol. 30, no. 2, pp. 54–60, 2013.

[13] A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The jkind model checker," in *Computer Aided Verification*, 2018.

[14] L. D. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[16] S. Nason and J. Laird, "Soar-rl: Integrating reinforcement learning with soar," *Cognitive Systems Research*, vol. 6, pp. 51–59, 03 2005.

[17] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *CAV*, 2014, pp. 334–342.

[18] T. Parr, *The definitive antlr 4 reference*. The Pragmatic Bookshelf, 2014.