Evaluating LTL Formulas for On-Board Unmanned Vehicle Health Monitoring

Michael Poteat
Modeling, Simulation, and Visualization Engineering
Old Dominion University
Norfolk, VA, United States
Email: me@mpote.at

Yiannis Papelis
Virginia Modeling, Analysis and Simulation Center
Old Dominion University
Suffolk, VA, United States
Email: ypapelis@odu.edu

***Abstract*—The proliferation of unmanned vehicle technologies has drastically increased their use in multiple domains. In the maritime domain, unmanned surface vehicles often pose special requirements for on-board health monitoring and fault mitigation due to long endurance, which increases the likelihood of failures when operating without human oversight. Whereas such vehicles can be equipped with numerous on-board sensors, detecting actual or impending failures is often more complicated than simply thresholding values of a sensor reading. In this paper, we will consider the use of Linear Temporal Logic (LTL) as a means to specify and then evaluate in real-time, the health status of an unmanned surface vehicle. This is accomplished by capturing nominal conditions in LTL formulas and then evaluating these formulas in real-time. The advantage of LTL is that it allows capturing value-based as well as time-based expectations for sensor readings when evaluating system status. We define a formal language which is an extension of LTL, and a corresponding software evaluation method with bounded performance. An example demonstration of the feasibility of the process is presented.**

Keywords–Linear Temporal Logic; safety; logic; model checking

I. Introduction

Monitoring the on-board status of unmanned maritime vehicles can prove challenging, for many reasons. Maritime vehicles operate in a relatively difficult environment in which debris, water spray, corrosion and other factors can degrade the performance of the system in ways that are not always immediately apparent. Furthermore, the use of automatic controllers often hides the onset of problems by compensating for such errors. Addressing such issues is often done by installing sensors that monitor for error conditions; however, there are difficulties in properly interpreting their readings. For example, consider an engine temperature gauge with a pre-set maximum safe limit. Shutting down the system based on that reading alone runs the risk of making an incorrect choice should the gauge itself fail and provide erroneous readings. A different but equally problematic scenario is gauge failure that displays a nominal temperature even though the actual temperature exceeds the safe limit. Because of the propensity of individual sensor failures, it is necessary the cross reference multiple sensor readings over time before making a determination of a fault. When under human supervision, sensor information is typically aggregated by the vehicle and transmitted to a monitoring/control station that displays all sensor readings, pushing the responsibility for making fault assessments and evaluating mission readiness to the human operator. Not only is this a difficult task for a human, but it is not transferable to unsupervised operations during which a vehicle must be able to make a determination of its ability to accomplish its mission autonomously.

Our approach is based on using Linear Temporal Logic (LTL) formulas Section III as a means of capturing nominal performance of the overall system. One advantage of LTL over other approaches is that LTL can capture the element of time in addition to fixed-in-time reading. Use of LTL formulas hence allows evaluating the behavior of the system over time and assessing if it operates within nominal parameters based on richer information when compared to point-in-time sensor readings. A key contribution of the paper is an efficient approach to evaluating the LTL formulas allowing their evaluation to be performed on-board the unmanned vessel.

The remaining of the paper is organized as follows: Section II overviews related work, Section III describes the LTL formalism in general and the specific portion used in our proposed system. Section IV outlines the method by which LTL formulas are evaluated in real-time based on sensor readings. Section V presents a test case of using LTL formulas to identify a nuanced failure in a maritime unmanned system and Section VI concludes.

II. Related Work

LTL has been used to model correctness properties of low-level software programs [1] and robotic motion planning [2] [3]. Safety properties can be falsified but not proved in general, as in sufficiently complex systems these statements are undecidable [4]. In practice, this is avoided by using a *bounded* variant of LTL known as metric temporal logic. This has been used to find the trajectory of safety properties over time [5]. Effectively, the decidability problem is avoided by evaluating safety expressions on a single system trajectory (i.e., a *trace*) in real-time.

On-board fault detection is also accomplished by using Bayesian networks [6], which consider the probability of the actual fault event as well as the reliability of the sensor, and thus makes a probabilistic estimate of a fault based on multiple sensor readings. One drawback is that a priori probability estimates of faults are needed, as well as relative sensor reliability weightings.

Another common approach for safety monitoring is the "residual method", whereby a real-time simulated version of the system is compared against the real system [7]. The residual (e.g., squared error) between the simulated and real system is computed; if the residual is too large, this indicates that some non-optimal state has been reached. A drawback of the residual approach is that you must model the system, and any errors in doing so, whether arising from system complexity or computational difficulty, may lead to false-positives. As well, it is not obvious how one would in general detect the exact problem that has occured solely from the residual, and indeed that problem has been an area of active research.

Our implementation is an alternative to both the residual and Bayesian methods, whereby one explicitly specifies invariants of how system variables must temporally relate to one another. In this "LTL approach", an explicit system model or simulation is not necessary, allowing it to be applied to systems intractable or uneconomic to explicitly model.

III. LTL Overview

LTL is essentially a generalization of Boolean logic, which adds a capability to model "propositions whose truth or falsity may depend on time" [8]. Practically, this adds a number of operators which specify temporal relationships between propositions. One way of thinking about temporal logic is that unlike first-order logic, it operates on countably infinite ordered sets [9] (i.e., sequences). In convention with the literature, we use the term *trace* to refer to these sequences, and the term *finite trace* when the sequence in question has finite size.

Since LTL is a generalization of Boolean logic, it inherits by default all of the common logical connectives intrinsic to that logic, which are enumerated by Table I. The so-called "application syntax" refers to the form used in application, chosen due to programming convention, as opposed to the symbolic form used in presentation of this document.

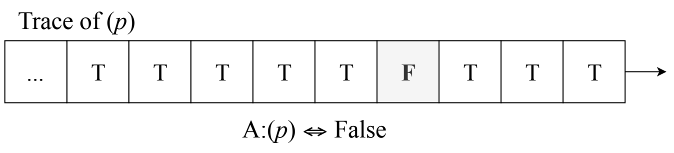
TABLE I. COMMON LOGICAL CONNECTIVES

Operator	*Symbol*	*Application Syntax*
Not	$\neg p$	`!p`
And	$p \wedge q$	`p && q`
Or	$p \vee q$	`p || q`

In that predicate logic models quantification (e.g., $\forall$, $\exists$), temporal logic models temporal relationships (e.g., A:, E:).The best way to understand the LTL formalism is by example. The arguably simplest temporal operation is A:(p), such that p is an arbitrary boolean proposition with values across time (i.e., a trace). This expression is read as "always", and simply specifies that p is always true across the trace. Figure 1 shows a case where A:(p) evaluates to true or false, respectively. In the first example, A:(p) evaluates to false because there exists a time in the past where p was false. In the second example, A:(q) evaluates to true for the opposite reason.

The second most basic operator is E:(p), which is the dual of A:(p). It is read as "eventually", and has much the same mathematical meaning: E:(p) is true if p was true at any point in time. These two operators are dual due to the relation $\neg$A:(p) $\Leftrightarrow$ E:$(\neg p)$. This should make intuitive sense: p was not always true if, and only if, there was a point at which p was false. That operator relationship is a salient similarity between temporal and first-order logic. In much the same way, Figure 2 exemplifies a set of traces where E:(p) evaluates to true or false, respectively.

The full set of temporal operators considered in our application is described in Table II. In addition to the ones already described, there exist three other operators whose illustrated explanation we will omit.



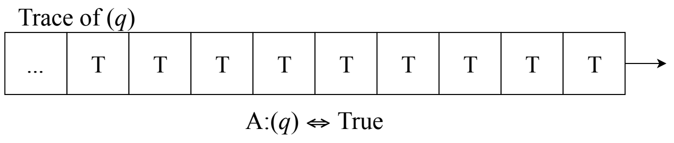
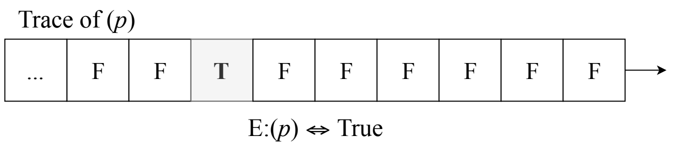


Figure 1. Always Operator on Two Example Traces.



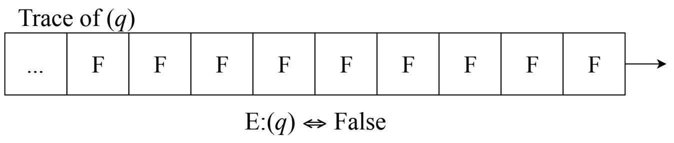


Figure 2. Eventually Operator on Two Example Traces.

TABLE II. LISTING OF TEMPORAL OPERATORS

Operator	*Syntax*	*Description*
Next	N: p	p was true one time-step ago.
Always	A: p	p was always true.
Eventually	E: p	p was eventually true at some point.
Until	p U: q	p was true up until just before q was true.
Release	p R: q	q was true up until p was true, after which q was false.

A. Introduction of Boolean Combinations

The true expressiveness of LTL arises from the ability to nest logical connectives and temporal operators in arbitrary ways. Referring back to Figure 1, the result of A:$(p \vee q)$ is true, because for every false entry in the trace of p, there exists a corresponding q entry that is true at the same time. The expression acted upon by a temporal operator may be any LTL expression, including other temporal operators. For example, in Figure 3, we see the intermediate values involved in the expression E:(A:(p)) operating upon a finite trace. A finite trace for this example was chosen only to keep the example simple; it works for general traces equivalently.

The key to understanding this nested temporal expression is that each temporal operator possesses its own trace (Boolean values through time), corresponding to whether or not the subsequence formed by taking the past at each point would result in a true value. However, the proper result of a temporal expression is the last value of the trace; This is indicated by the bottom arrow in Figure 3. A conceptual justification for this is that the last value of the trace is what the value is "now".

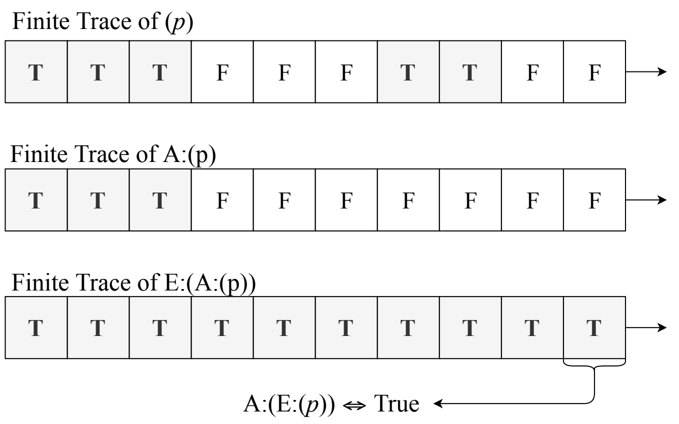


Figure 3. Nested Operator on Finite Trace.

It is important to note that this interpretation is not unique; it is equally as valid to take the first value of the trace as “now” and consider all other trace values to be future values.

B. Introduction of Metric Temporal Logic

A common way to introduce real-time in the (LTL) syntax is by “replacing the unrestricted time operators by time-bounded versions” [10]. This allows for temporal operators to factor a metric of time, and to have essentially a time-bounded range of concern. In addition to making the logic much more expressive, it also has important considerations for real-time evaluation of the logic for practical applications.

We utilize MTL operators by defining some slightly modified syntax. For example, the time-bounded version of “always” is A:$^t_s(p)$ where $s, t \in \mathbb{Z}^{\geq}$. In our formulation, the 0th entry corresponds to “now”, and all other entries incrementally refer to past values. Table III defines the application syntax and mathematical symbology used to denote the metric temporal operators.

TABLE III. SYNTAX OF METRIC TEMPORAL OPERATORS

Operator	*Symbol*	*Application Syntax*
Always	A:$^t_s(p)$	`A:s:t,(p)`
Eventually	E:$^t_s(p)$	`E:s:t,(p)`
Until	pU:$^t_s q$	`p U:s:t, q`
Release	pR:$^t_s q$	`p R:s:t, q`

C. Introduction of State Variables

The final extensions we include in our formalism are basic arithmetic and relational operations, as well as numerical state variables. State variables are real numbers which possess a real-valued trace (i.e., values through time). Effectively, this allows us to construct expressions which model the temporal relationship of real-valued variables through time. These values can represent sensor readings or other on-board state varibles. A comprehensive example of this capability is illustrated by Figure 4, which shows the intermediate values associated with evaluating E:(A:$^2_0(x > y)$), given that $x, y \in \mathbb{R}$. This expression is equivalent to the existence of a three-unit contiguous time region during which x is larger than y.

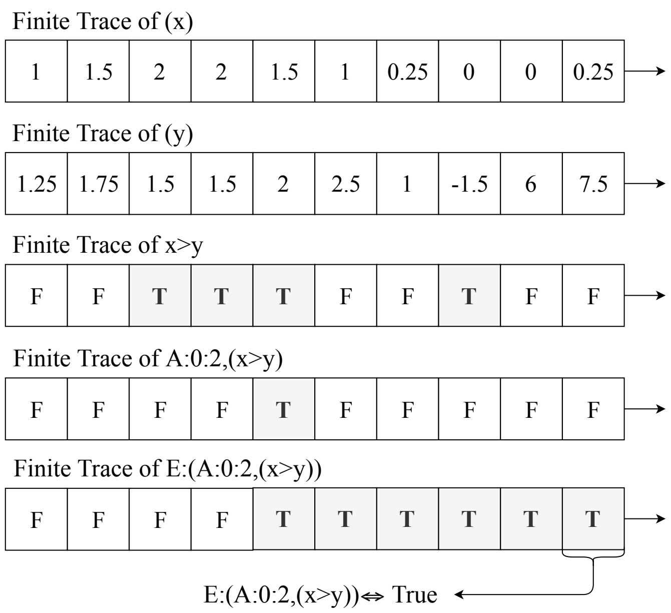


Figure 4. Bounded Temporal Expression on State Variables.

IV. METHOD

Real-time evaluation of the LTL formalism described by Section III was implemented via a Robot Operating System (ROS) package developed for the purpose. ROS is a commonly used middleware for developing robotics applications, providing algorithms and visualization tools. Some of the basic functionality provided is message-passing, which is implemented through so-called “ROS topics”, and serves as the basis of the application.

Given user-specified LTL formulas and associated topic names, the program parses and evaluates the formulas, publishing the truth result to the rest of the ROS system. The flow of inputs and outputs involved in this process is illustrated in Figure 5. Dotted arrows represent external interfaces to the system.

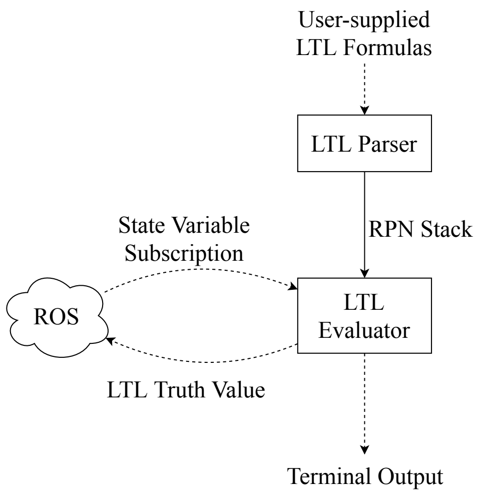


Figure 5. System Diagram of LTL Parser and Evaluator.

The evaluation of an LTL expression occurs in two steps, as illustrated in Figure 6. Of note is that explicit construction of the abstract-syntax tree is not performed during parsing, but is provided here only for explanation.

1) The expression is parsed while converting from infix to RPN.
2) Variable tokens are replaced with the corresponding data, and the stack machine is executed.

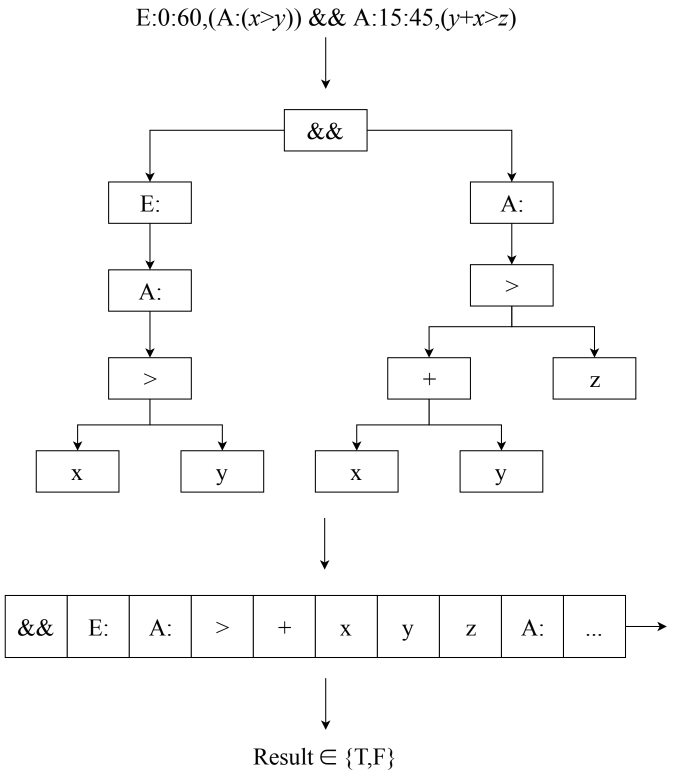


Figure 6. Generation of RPN Stack Machine.

Parsing is performed via Dijkstra's shunting-yard algorithm, converting the expression string to an array of Reverse Polish Notation (RPN) tokens. The evaluator then, on each arrival of new data, evaluates the RPN expression using a stack-based postfix evaluation algorithm. Each element of the stack is a trace, and each operator is a function of one or more traces.

When a real number is encountered in the execution, it is interpreted as a trace consisting solely of that real number. Hence, when a variable is introduced and compared to that number, the result is a boolean trace representing the result of that comparison over time. For all arithmetic, relational, and logical operators the result is simply that operator applied pair-wise to each element of the two corresponding traces. In the case of 1-arity operators e.g., $!p$, the operator is simply applied to each element of the trace. For temporal operators, each trace element's result is a function of the previous elements in the same way as described in Section III. Each element of every trace is of real type, and is automatically type cast depending on the operator.

The internal procedure used to calculate the 'Always' operator is described by Algorithm 1. If a boundedness operator

Algorithm 1: 'Always' operator. Resultant array is true up until x_i is false.

```
1 function A ($x$);
  Input  : Array of reals $x$ of size $n$
  Output: Array of reals, size $n$
2 boolean : $\alpha = $ true;
3 $\alpha = $ false;
4 for $x_i \in x$ do
5     $\alpha = x_i \wedge \alpha$;
6     $x_i = \mathbb{R}(\alpha)$;
7 end
8 return $x$;
```

is applied the procedure described by Algorithm 1 will operate only on a contiguous subsequence of the input trace. All non-zero real numbers (approximated by floating point) are type cast to `true` if acted upon by a boolean operator.

We achieve bounded performance by allowing the user to specify the maximum trace size for each LTL formula. Once the trace reaches that size, all LTL formulas, whether they are ultimately bounded or not will only act upon data within the specified time window of the current time. This is to ensure that the computation required to evaluate a given LTL formula is sublinear with respect to the current time of operation. One future work considered is automatic generation of an appropriate maximum trace size given an LTL formula.

V. RESULTS

This method was applied to an autonomous sea vessel in order to detect motor misalignment conditions, which occur when there exists an offset between the steering control value and the angle between the vessel and one, or both, of the motors. The control interface is a four-vector, with each element controlling the rotation (with respect to the craft) and effort of each of the two motors respectively. We use the term "effort" a percentage of the total power available to the system for acceleration that abstracts away physical details. For context, a simplified diagram of the physical placement of the motors can be seen in Figure 7. The controls values are subject to the constraints defined by (1) and (2).

$$-100 \geq E_N \leq 100 \tag{1}$$

$$-90^\circ \geq \theta_N \leq 90^\circ \tag{2}$$

The method was implemented as a configurable ROS component, which subscribes to the topics necessary to detect the suboptimal condition. The implementation allows for multiple LTL formulas each corresponding to a set of ROS topics. (3) specifies the LTL formula written to detect the motor misalignment condition, where $|x|$ specifies the absolute value operation.

$$\text{A:0:60,}(E_L = E_R \wedge |S_L| < 3 \wedge |S_R| < 3 \wedge V_{yaw} > 0.3) \tag{3}$$

We may break (3) into three logical clauses. If the following three conditions are true for the last sixty time units, the motor is misaligned.

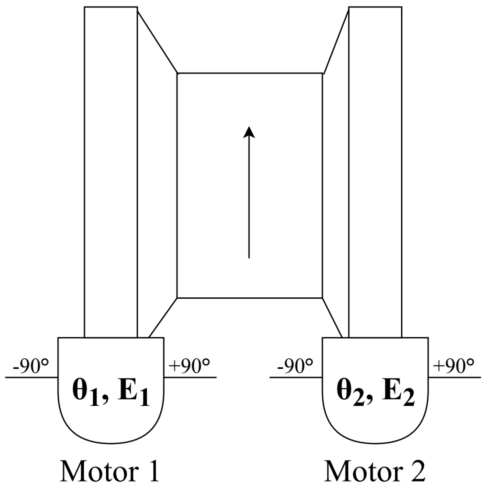


Figure 7. Diagram of Autonomous Vessel in Overhead View.

1) The efforts from both motors are equal.
2) The steering angles from both motors are less than 3 degrees, and...
3) The yaw-velocity of the craft is above a certain threshold.

The LTL formula was applied to two simulations of the autonomous vessel, each under an equivalent control trajectory. In the first (control) simulation, both motors are correctly aligned. In the second, the left motor is misaligned by 10°. The resultant angular velocity of both 10-second simulations is illustrated by Figure 8. The difference in yaw velocity, and particularly the spike at 2 seconds in the misaligned case can be attributed to the alignment discrepancy.

The control trajectory applied was generated by a simple driver code. For 2 seconds, an effort value of 50 and 100 was applied to the left and right motors respectively. Then, for the next 2 seconds, an effort value of 100 was applied to both motors for 2 seconds. Finally, the vehicle was allowed to coast under no effort. The control trajectory was chosen to be deterministic and not unlikely to occur during user operation.

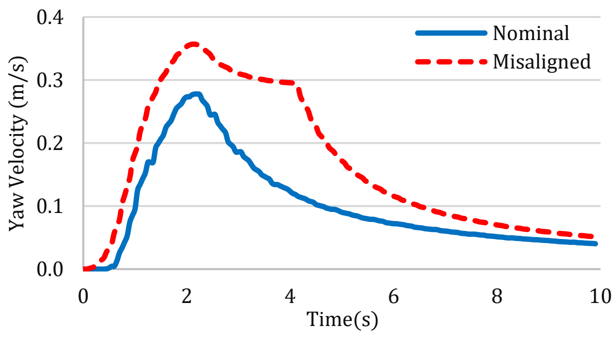


Figure 8. Angular Velocity of Nominal vs. Misaligned Case.

The resultant boolean signal for both the nominal case and misaligned case is seen in Figure 9. Due to the to the structure of the LTL formula used, there is a delay present in the result. It is possible to decrease the delay, but at risk of causing false-positives. As in most forms of signal processing, a tradeoff in delay and accuracy is present.

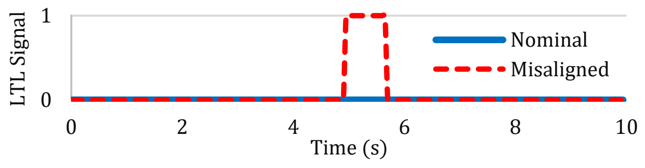


Figure 9. LTL Signal of Nominal vs. Misaligned Case.

Instead of integrating the squared error between what is expected to happen (via a physics model) with what was observed, i.e., the "residual method", we write specific temporal scenarios, which would only occur if the condition is present. The LTL approach allows us to both be more specific about error detection, and can be deployed in scenarios where an accurate physics model is intractable to compute.

VI. Conclusion and Future Work

Autonomous systems require ways to detect, interpret, and even anticipate problems. On-board sensors provide information, but unless there is a trivial interpretation (i.e., battery voltage dropping below a threshold), it is difficult to make a singular assessment about on-board status based on several sensor readings. Furthermore, proper interpretation of sensors cannot be done only for a single time, but must be done over a history. Proper sensor values may be temporally correlated in non-trivial ways. LTL is a convenient formalism for capturing the expected behavior of the system via mathematical modeling. Failures in the system can be directly inferred from evaluating these LTL expressions.

In this paper, we have demonstrated a practical LTL evaluation method that has bounded performance with respect to temporal formula evaluation at given time-points. For tele-operated systems, we view this as a compression scheme to encode high-dimensional temporal data into a form parsable by a human operator controlling the system. For on-board systems, this method effectively addresses the problem of autonomously capturing the nominal performance of the overall system.

There are three major directions planned for future work of this method. Currently, maximum trace size is manually set in order to provide bounded performance. However, it seems possible to automatically derive the maximum trace size from a provided LTL formula. As well, memoization of certain intermediary values may improve on the computational complexity of the method. Finally, a large part of the effort expended in using this method is coming up with a LTL formula that captures the desired behavior. Supervised machine learning techniques could be used to automatically generate an appropriate LTL formula given examples of nominal and "problem" mission trajectories, enabling rapid development of status assessment systems.

References

[1] J. Morse, L. Cordeiro, D. Nicole, and B. Fischer, "B.: Context-bounded model checking of LTL properties for ansi-c software," in *SEFM 2011. LNCS.* Springer, 2011, pp. 302–317.

[2] G. E. Fainekos and H. Kress-gazit, "Temporal logic motion planning for mobile robots," in *In Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2020–2025.

[3] A. Albore and P. Bertoli, “Safe ltl assumption-based planning,” in *In Proc. 16th Int. Conf. on Planning and Scheduling (ICAPS-06*.

[4] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, “What’s decidable about hybrid automata?” in *Journal of Computer and System Sciences*. ACM Press, 1995, pp. 373–382.

[5] J. Schumann, K. Y. Rozier, T. Reinbacher, O. J. Mengshoel, T. Mbaya, E. Al, J. Schumann, K. Y. Rozier, T. Reinbacher, O. J. Mengshoel, and T. Mbaya, “Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems,” in *in Proceedings of the 2013 Annual Conference of the Prognostics and Health Management Society (PHM2013*, 2013.

[6] R. Zhang, X. Hu, H. Wang, and H. Yao, “Fault diagnosis method in complex system using bayesian networks sensitivity analysis,” *Information Technology Journal*, vol. 14, pp. 24–30, 01 2015.

[7] R. Reiter, “A theory of diagnosis from first principles,” 1987.

[8] F. KrÃűger, *Temporal Logic of Programs*, ser. EATCS Monographs on Theoretical Computer Science. Springer, 1987, vol. 8.

[9] E. A. Emerson, “Temporal and modal logic,” in *Handbook of Theoretical Computer Science*. Elsevier, 1995, pp. 995–1072.

[10] R. Alur and T. A. Henzinger, “Logics and models of real time: A survey,” 1992.