Proceedings of the ASME 2020
International Mechanical Engineering Congress and Exposition
IMECE2020
November 15-19, 2020, Portland, OR, USA

IMECE2020-23234

FORMALIZING PERFORMANCE EVALUATION OF MOBILE MANIPULATOR ROBOTS USING CTML

Omar Aboul-Enein*
Salisbury University
Salisbury, Maryland
National Institute of Standards
and Technology
Gaithersburg, Maryland
Email: omar.aboul-enein@nist.gov

Yaping Jing
Salisbury University
Salisbury, Maryland

Roger Bostelman
National Institute of Standards
and Technology
Gaithersburg, Maryland

ABSTRACT

Computation Tree Measurement Language (CTML) is a newly developed formal language that offers simultaneous model verification and performance evaluation measures. While the theory behind CTML has been established, the language has yet to be tested on a practical example. In this work, we wish to demonstrate the utility of CTML when applied to a real-world application based in manufacturing. Mobile manipulators may enable more flexible, dynamic workflows within industry. Therefore, an artifact-based performance measurement test method for mobile manipulator robots developed at the National Institute of Standards and Technology was selected for evaluation. Contributions of this work include the modeling of robot tasks implemented for the performance measurement test using Petri nets, as well as the formulation and execution of sample queries using CTML. To compare the numerical results, query support, ease of implementation, and empirical runtime of CTML to other temporal logics in such applications, the queries were re-formulated and evaluated using the PRISM Model Checker. Finally, a discussion is included that considers future extensions of this work, relative to other existing research, that could potentially enable the integration of CTML with Systems Modeling Language (SysML) and Product Life-cycle Management (PLM) software solutions.

*Address all correspondence to this author.

1 INTRODUCTION

In the field of formal verification, Computation Tree Measurement Language (CTML) offers many advantages over conventional methods for conducting performance-reliability analysis. CTML combines performance and reliability evaluation capabilities under one language [1]. Jing and Miner establish the theoretical foundation for CTML, describe the advantages of the language, and provide an example of the language usage through modeling the Dining Philosophers problem. In their work, a few primary advantages of CTML are noted. First, CTML is able to respond to nested queries with either a real-valued quantity or a probability. In addition, CTML matches the functionality of Probabilistic Computation Tree Logic (PCTL) [2] and can respond to a (non-trivial) subset of Probabilistic Linear Temporal Logic (PLTL) queries that are not expressible in PCTL [1, 3]. Most importantly, the functionality of CTML extends beyond the aforementioned logics as the language can answer queries not expressible in either PCTL or PLTL [1]. For example, CTML can answer survivability queries, which ask how much time remains until an event occurs given that another event has already occurred. Finally, Jing and Miner theoretically established the running efficiency of CTML as polynomial with respect to the both the formula and state size [1].

The theoretical advantages of CTML just described also suggest potential benefit towards Industry 4.0 applications. Industry 4.0 is the convergence of "robotics, cyber-physical systems,

software services, and human participants" towards "interoperability, information transparency, technical assistance, and decentralized decisions" [4]. For example, consider the following work in assessing picker robot workflow (modeled using workflow nets) conformance to the specifications of an automated warehouse [4]. CTL queries were formulated to verify safety properties of the workflow, such as an absence of deadlocks, proper workflow completion, and proper workflow termination among other properties. CTML could theoretically extend this assessment to include how much time remains until a product is shipped given the robot just picked up the product or the probability a deadlock occurs within 15 minutes given that a procurement has just completed [1,4].

While the theoretical advantages of CTML have been tested through a dining philosopher example scenario and the original state-based language has been extended to support reasoning over paths with multiple actions and states, the adoption of CTML towards Industry 4.0 applications would benefit from a more focused exploration of its use in a real-world problem [1,5]. Therefore, in this work, we explore the viability of CTML and its theoretically established benefits by applying the language towards the formal verification of a contemporary, manufacturing-based application. The application consists of a performance measurement test method for mobile manipulators currently under development at the National Institute of Standards and Technology (NIST)[1] [6]. Mobile manipulation offers an abundance of potential real-world uses, especially within manufacturing environments. These applications include tasks such as sanding or painting large surfaces, welding large parts, managing conveyors, and other forms of flexible manufacturing [7]. It should also be noted that existing work has applied Model-Based Systems Engineering (MBSE) methodologies to the performance measurement test using Systems Modeling Language (SysML) [8]. Whereas the authors state the SysML model was verified through a systems review and referenced experiments, the utilization of CTML exemplifies simultaneous, probabilistic verification and performance evaluation of the test method procedure [8].

To formally evaluate the performance measurement test method, we selected the Petri-net formalism described in [9] to develop an initial, high-level model of the robot tasks performed as part of the mobile manipulator performance test method. We also selected and formulated a series of sample queries to be answered by CTML. We then adapted the model and queries for use with another existing tool, the Probabilistic Symbolic Model-Checker (PRISM) (Download at: `http://www.prismmodelchecker.org/download.php`) [10]. Following this step, we compared the supported queries, numerical results, Central Processing Unit (CPU) runtime, and ease of implementation. Finally, given the findings of the study, we discuss potential future applications of CTML, which includes integration with the previously mentioned SysML model of the performance test method and integration with Product Life-cycle Management (PLM) software.

[1]Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

2 BACKGROUND

2.1 About the Petri net Formalism

The Petri net formalism was selected to develop the initial model both for the high-level convenience of the formalism and for its ability to be procedurally converted to a format usable by CTML and PRISM. Petri nets are described as a "bipartite, directed graph populated by three types of objects" [9]. The three objects include places (graphically represented as hollow circles), transitions (represented by boxes or solid lines), and directed arcs. Places represent a possible state, condition, or available resource and transitions represent a change of state or action. Additionally, tokens (graphically represented as one or more filled circles or a variable on a place) represent the current fulfillment of a condition or availability of a resource. A "marking" consists of the distribution of tokens to places at any given time (with the "initial marking" at time zero). Arcs, which may be weighted, may only connect places to transitions or vice versa. Upon "firing" a transition, the number of tokens required by the weight of the input arc is consumed from the input place and the number of tokens indicated by the weight of the output arc is produced on the output place. A transition can only be fired, or is "enabled", if the input place contains enough tokens to satisfy the weight of the input arc. Inhibitor arcs (marked by a circle instead of an arrowhead) instead require that places have no tokens in order to fire [9].

2.2 The Mobile Manipulator Performance Test Method

The performance measurement test method modeled in this paper utilizes a Re-configurable Mobile Manipulator Artifact (RMMA) [6]. The RMMA provides a known, user-adjustable measurement uncertainty to compare to ground-truth. Additionally, The RMMA consists of an anodized, machined-aluminum build and has an adjustable height and table surface rotation. The table surface can be drilled and tapped with holes arranged in various geometric patterns (such as a square, circle, or sinusoid). The holes are analogous to points mounted with retro-reflective targets, which are digitally detected through the use of a retro-reflective laser sensor/emitter (RLS) mounted on the end-effector of a robotic arm.

The use of this non-contact registration sensor reduces the risk of collisions and is comparable to operations such as the classic peg-in-hole assembly task [11]. An example of an RMMA, RLS, and retro-reflective target is shown in Fig. 1a. The

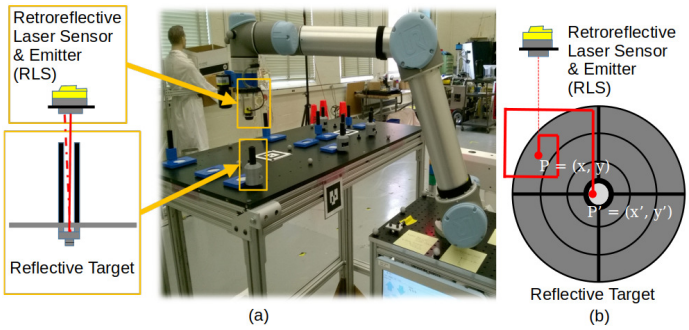

FIGURE 1. (a) THE RETRO-REFLECTIVE LASER SENSOR AND EMITTER (RLS) USED FOR ASSEMBLY POINT DETECTION. (b) TOP VIEW OF RLS AND SPIRAL SEARCH PATTERN

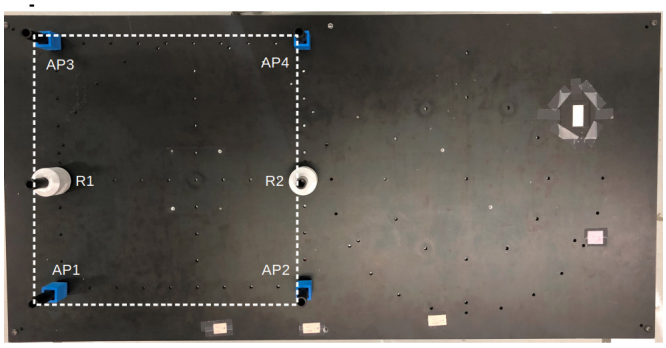

FIGURE 2. EXAMPLE OF RMMA CONFIGURATION WITH AP REFLECTORS IN A SQUARE PATTERN.

description of the general performance measurement test specifies two potential cases for manipulator and vehicle coordination. These levels of coordination are described as "indexed" (the manipulator performs assembly while vehicle is stopped) and "dynamic" (both the manipulator and the vehicle are in motion) [6]. The primary metrics of interest include a pass/fail test for initially detecting the retro-reflective targets at the assembly points, and a repeatability test in positioning the manipulator over the assembly points [6]. The RMMA can also evaluate and compare coordinate registration methods, and the time taken to locate the retro-reflective targets is also of interest for these methods [12].

3 PETRI NET MODEL DESCRIPTION

3.1 Model Scenario and Parameters

We now specify the instance of the test scenario to be modeled. First, we chose to focus on modeling a mobile manipulator utilizing an Automatic Guided Vehicle (AGV), specifically one that cannot re-compute paths around unexpected obstacles. With this system, an industrial manipulator arm is also fixtured onboard the AGV throughout the assembly tasks. Second is that we chose to model the indexed case of the mobile manipulator performance test instead of the dynamic case (see Section 2.2). Finally, we assume the RMMA is configured with six retro-reflective targets arranged in a square as shown in Fig. 2.

For the performance test method, the AGV travels to and parks at a sequence of n arbitrary locations next to the RMMA, which are referred to as "stops". Each stop is assumed to be unique in either position and/or orientation. While in transit, it is possible that an obstacle or hazard causes the AGV to emergency stop (e-stop). Once the obstacle has been removed, the operator can clear the e-stop, allowing the AGV to resume the test. The AGV is allowed to e-stop E times before the entire test is aborted. While the AGV is parked at a stop, the manipulator arm first performs coordinate registration of the mobile manipulator to the RMMA by locating two retro-reflective targets, denoted R1 and R2. The manipulator arm then locates the other four retro-reflective targets, which are used as verification points to test the accuracy of the initial coordinate registration and manipulator positioning. These reflectors are denoted as assembly points 1 (AP1) through 4 (AP4). For both previous steps, the same target localization method is used, which consists of the manipulator tracing a square spiral pattern as visualized in Fig. 1b [12]. The spiral search is allowed to execute t steps in the localization pattern before the search is aborted. The detection of each assembly point, as part of the verification phase, can be repeated r times. Finally, this entire process can be repeated for N runs of the test. The values, or parameters, are adjustable settings of the performance test method. Changing each value as input to the model can increase or decrease the total number of reachable states associated with the Petri net.

3.2 Description of the Petri net Model

The model and initial marking of the mobile manipulator performance test (divided into five subnets) is shown in Fig. 3 - 7. Places are denoted P_i, where i is an integer subscript, and transitions are denoted t_j, where j is an integer subscript. Subnets, shown as rectangles, are annotated to indicate the connections between subnets. The exponential distribution was chosen as the firing distribution for transitions. The rate parameter of the exponential distribution was taken to be inversely proportional to the number of tokens currently in the Petri net at a given time.

3.2.1 AGV Subnet. The first subnet, shown in Fig. 3, defines the behavior of the AGV during the test. The initial token on P_1 represents the vehicle parked at the initial home position. After leaving the home position, the AGV proceeds through the sequence of n stops next to the RMMA. As mentioned previously, the AGV may e-stop while in transit (see Section 3.2.2). A token on P_2 indicates the AGV is finishing the current test run. P_3 maintains a count of the remaining number of test runs. P_3 initially has N tokens, which decreases every time t_2 is fired. For

each run, the AGV parks at each of the n stops in sequence (see Section 3.2.3). When the token count on P_3 reaches 0, t_3 must fire, and the AGV attempts to return to the home position. A token on P_4 indicates the AGV has reached the home position and completed the test.

3.2.2 E-Stop Subnet. The subnet modeling the occurrence of an AGV e-stop is shown in Fig. 4. P_5 represents the vehicle en-route to a stop at the RMMA. P_6 is initialized with E tokens and counts the number of remaining e-stops allowed. P_7 counts the number of e-stops that have occurred previously. A token on P_8 indicates the vehicle is currently e-stopped, with an obstacle preventing the vehicle from currently reaching the intended destination. Since it is possible for the vehicle to recover from the e-stopped state, t_6 is included to connect P_5 to P_8.

3.2.3 Stop Subnet. The subnet modeling the actions of the AGV for each stop is shown in Fig. 5. A token on P_9 represents the AGV sending a signal to the manipulator to begin coordinate registration to the RMMA. P_{10} counts the number of times the AGV has previously visited the stop. A token on P_{11} indicates the AGV is currently parked at the RMMA. Transition t_8 passes the signal from the AGV to the manipulator and initializes r tokens on P_{18} in the manipulator subnet. t_9 passes a signal from the manipulator to the AGV indicating the manipulator has finished coordinate registration and verification. Upon receiving this token, the vehicle can proceed to the next stop.

3.2.4 Manipulator Subnet. The subnet defining the behavior of the manipulator arm is shown in Fig. 6. A token on P_{12} indicates the manipulator has received the signal from the AGV to begin coordinate registration. The manipulator begins in a stowed position, which allows for safe transport, and is represented by a token on P_{13}. Upon receiving the signal from the AGV, the manipulator moves to a stage position whereby it can access the RMMA. A token on P_{14} indicates that the manipulator is moving to the stage position. P_{15} and, similarly, P_{20} represent an e-stopped state. A token on P_{16} indicates the manipulator successfully reached the stage position. The manipulator then attempts the localization of two reflectors, R1 and R2, to complete coordinate registration. If either point cannot be found (as indicated by t_{26} in the spiral search subnet), the verification phase is skipped. If both points are localized, the manipulator proceeds to search for four verification reflectors, AP1 through AP4. A token on P_{17} indicates the end of a verification loop. P_{18} counts the remaining number of times that the verification loop will be repeated. A token on P_{19} represents the manipulator attempting to reach the stow position again. A token on P_{21} represents a signal being sent back to the AGV to proceed to the next stop.

3.2.5 Spiral Search Subnet. The subnet for modeling the spiral search localization method is pictured in Fig. 7, with Fig. 1b providing a visualization of the search routine itself. For the spiral search, a token on P_{22} indicates the manipulator in motion to position the RLS over an initial search point on the RMMA. The presence of a token on P_{23} represents a failure by the manipulator to position the toolpoint over the initial search point, which results in an e-stop. This is similarly the case for P_{30}. A token on P_{24} indicates the successful positioning of the RLS over the initial search point. Upon this success, t tokens are placed on P_{25}. P_{25} counts the remaining steps allowed in the search pattern. Upon reaching the initial search point, the RLS either immediately detects the retro-reflective target or does not. The detection of the target is indicated by a token on P_{26} while failing to detect the retro-reflective target is indicated by a token on P_{27}. If the target is not detected and search steps remain, the manipulator increments along the search path in attempt to localize the retro-reflective target. A token on P_{28} indicates the manipulator attempting to position the RLS over the next step along the search path. P_{29} counts the number of search steps previously taken by the manipulator.

4 EXPERIMENTS

4.1 Methods

We begin by explaining the intermediate steps taken to prepare the model and queries as input for the software implementation of CTML. Starting with the Petri net, we used the Stochastic Model-checking Analyzer for Reliability and Timing (SMART) (Download at: `https://asminer.github.io/smart/`) tool to convert the Petri net expression of the model to a Discrete Time Markov Chain (DTMC) [13]. Next, we used a custom-written Java program to express the DTMC in the format required by the CTML software. The Java program also added a set of "atomic functions" needed by the chosen queries [1]. The atomic functions calculate a starting quantity based on a given DTMC state. The augmented DTMC is then fed as input to the CTML software tool, along with the desired queries, to obtain the numerical result for each query (either a probability or a quantity). For this study, a total of ten models were generated from the Petri net by repeating this process with different input parameters. The parameters were varied to test a variety of model sizes ranging from a couple hundred states to approximately 4 million states.

For the two CTML queries that were expressible by PRISM, we were interested in comparing the numerical results, query support, ease of implementation, and CPU time between the two tools. To prepare the PRISM input, we started with the same DTMCs converted from the Petri net models via SMART. Unlike the CTML software, the PRISM tool required deadlocked states in the DTMC to have self-looping arcs, which increased the number of arcs for the PRISM input model (see Tab. 1). Since PRISM supports the specification of reward functions, which map states

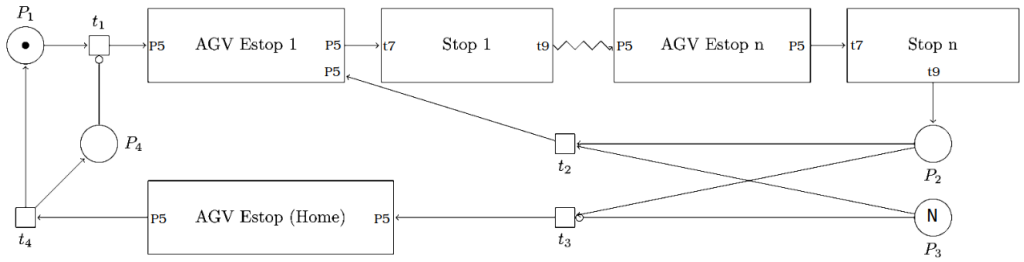

FIGURE 3. MOBILE MANIPULATOR PETRI NET: AGV SUBNET.

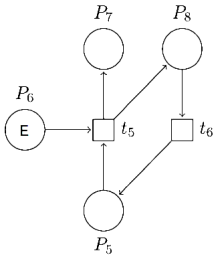

FIGURE 4. MOBILE MANIPULATOR PETRI NET: E-STOP SUBNET.

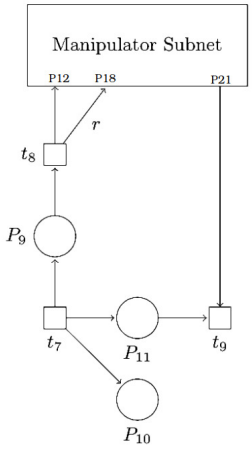

FIGURE 5. MOBILE MANIPULATOR PETRI NET: STOP SUBNET.

in the model to real values, each atomic function in CTML is redefined as a label (denoted by an "l" in its name) and a reward function, respectively [14]. Like CTML, the translated DTMC and queries can be fed into the PRISM tool as input.

The queries were tested on a computer with a 3.2GHz Intel Core i5 and 16GB of 1867MHz memory, and running MacOS X with the Java Development Kit (JDK) version 1.7. For CTML (a prototype tool) and PRISM (version 4.4), the runtime was benchmarked using the "User Time" field outputted by the `time` command line utility. This field is the total CPU time used by the tool excluding time used for executing operating system kernel code. The PRISM software tool also provided an internally measured time for model checking, which we called the "PRISM time" for short. To reduce the model-construction time in PRISM, we used "Explicit Model Files" and ran the queries using the explicit computation engine, which was specified using the `-ex` flag. With explicit model files, the states, transitions, and labels of the model were pre-constructed and each stored in a separate file. These models were then imported for model-checking by using the `-importmodel` flag. Additionally, to ensure there was enough memory for running the queries on the models, the maximum memory was expanded to 8 GB using the `-cuddmaxmem` and `-javamaxmem` flags [14]. All other PRISM configuration flags were kept to their default values.

4.2 Sample Queries

The sample queries were selected for their ability to test the boundaries of supported query types for existing temporal logics (as discussed in the Section 1) and for their relevance to the previously described metrics of the performance test method. Each query is presented in their English and CTML expressions. A complete explanation of CTML syntax and semantics is provided by Jing and Miner [1]. Likewise, for a complete explanation of the PRISM syntax and semantics, we refer to the PRISM manual [14]. Queries that could not be consistently translated into any of the logics supported by the PRISM tool are accompanied by an explanation. Otherwise, we present the query formulation in the appropriate logic. Finally, for all of the following queries, the atomic function *one* denotes a value of 1 for all states in the DTMC.

4.2.1 Query 1. *"What is the expected time until an e-stop occurs as the AGV attempts to park at stop 1?"* We define an atomic function *agvestop*1 with value 1 for the states where AGV is e-stopped, and 0 otherwise. Here, the atomic function *one* denotes one time unit per state. The query can be expressed

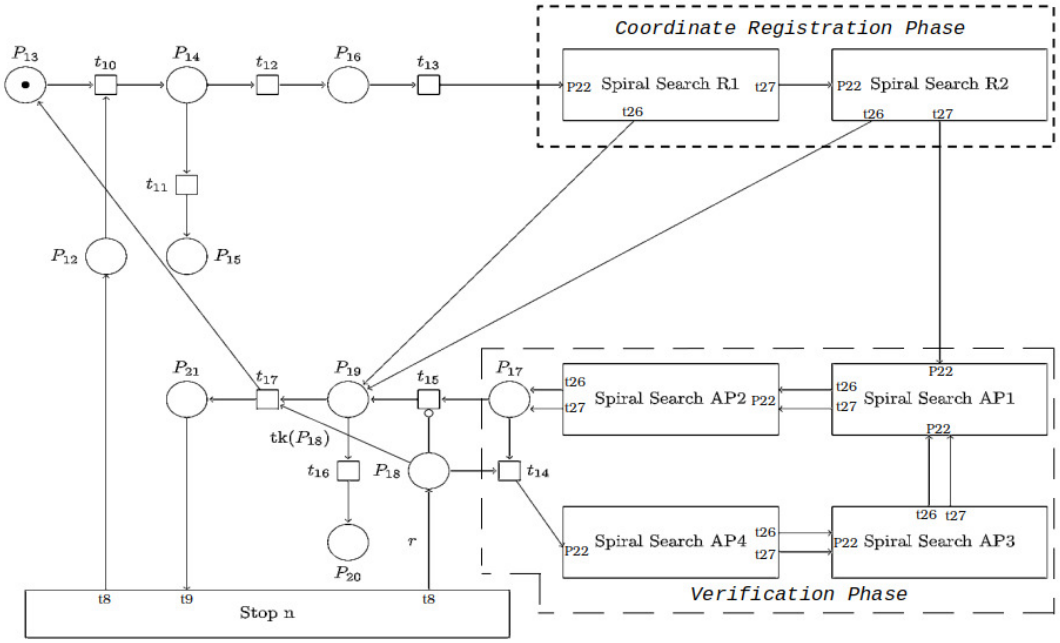

FIGURE 6. MOBILE MANIPULATOR PETRI NET: MANIPULATOR SUBNET.

in CTML as:

$$M\,one\,U_{+}\,agvestop1 \tag{1}$$

The Until Plus operator U_{+} accumulates *one* along a path before reaching the first $agvestop1$ state, s, with $agvestop1(s) > 0$. The M operator denotes the expected value (mean) over the path formula $one\,U_{+}\,agvestop1$, which captures all paths reaching $agvestop1$ states. Attempting to evaluate Query 1 in PRISM resulted in several limitations and inconsistencies. First, attempting to import the reward functions while using the explicit engine resulted in an error stating there was no model generator to construct the rewards structure. This limitation was addressed as of PRISM version 4.6, however, due to the recent timing of this release, we were unable to directly test the newly supported functionality [15]. Therefore, without the explicit engine, the long model construction time discussed previously made it impractical to evaluate Query 1 on the larger models. With smaller models, for which the model construction time was not prohibitive, evaluation was attempted using the hybrid engine. In this case, PRISM returned a result of infinity, which was inconsistent with CTML. This behavior is a design choice in the PRISM property specification language and occurs when the probability of reaching a destination state is less than one [14, 16]. By specification of the mobile manipulator test scenario, it is not guaranteed that the AGV will e-stop. Therefore, Query 1 could not be fully evaluated using PRISM.

4.2.2 Query 2. *"What is the probability that the manipulator eventually fails to locate AP1 while the AGV is parked at stop 1?"* For this query, we define an atomic function $ap1fail\text{-}stop1$ with value 1 for the states where the manipulator fails to locate AP1 while the AGV is parked at the first stop next to the RMMA, and 0 otherwise. This situation occurs if the maximum number of steps in the spiral search pattern has been exceeded and AP1 is still not detected by the RLS. This query can be expressed by CTML as:

$$M\,one\,U_{\times}\,ap1fail\text{-}stop1 \tag{2a}$$

This is similar to Query 1, except that here we use the Until Multiply operator $U_{\times}$ to compute a probability along a path before reaching the first $ap1fail\text{-}stop1$ state s with $ap1fail\text{-}stop1(s) > 0$. The expected value operator M over the path formula $one\,U_{\times}\,ap1fail\text{-}stop1$ computes the sum over all states s with $ap1fail\text{-}stop1(s) > 0$, the probability that s is reached before any other state s' with $ap1fail\text{-}stop1(s') > 0$, multiplied by $ap1fail\text{-}stop1(s)$. Note that the CTML formula for Query 2 is shown in Eqn. 2a. Query 2 can be successfully translated to PCTL in PRISM using Eqn. 2b. The reachability operator F, as described in the PRISM manual, can be used to express formulas like "$true\ U\ \mathrm{p}$", which specifies that p is eventually true. Then, the probability for the formula can be obtained via the operator $P_{=?}$ in PRISM.

$$P_{=?}\,(F\,ap1fail\text{-}stop1\_1) \tag{2b}$$

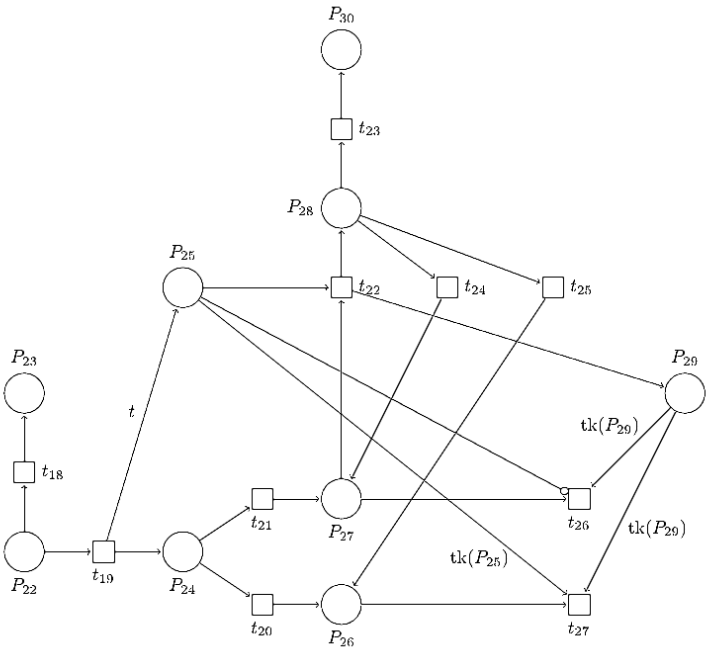

FIGURE 7. MOBILE MANIPULATOR PETRI NET: SPIRAL SEARCH SUBNET.

4.2.3 Query 3. *"Given that an AGV e-stop occurred while the AGV was attempting to reach stop 1, what is the probability that the manipulator fails to locate AP1?"* Similar to Query 6 presented by Jing and Miner for the dining philosopher model [1], Query 3 requires a conditional probability of the form $Pr(B|A)$, where A denotes an AGV e-stop while the AGV attempts to park at the first stop next to the RMMA. B denotes a failure to localize the AP1 reflector while the AGV is parked at the first stop next to the RMMA. To solve the conditional probability, we find the probability of events B and A occurring simultaneously and then divide the result by the probability that event A occurs. The CTML formula for this query is shown in Eqn. 3a, which can also be successfully translated to PLTL in PRISM and is shown in Eqn. 3b.

$$\frac{M\,one\,U_{\times}\,(\,(M\,one\,U_{\times}\,ap1\,fail\text{-}stop1\,)*agvestop1)}{M\,one\,U_{\times}\,agvestop1} \tag{3a}$$

$$\frac{P_{=?}\,(\,F\,(\,(agvestop1\_1\,)\wedge(F\,ap1\,fail\text{-}stop1\_1\,)\,)\,)}{P_{=?}\,(F\,agvestop1\_1\,)} \tag{3b}$$

4.2.4 Query 4. *"What is the expected number of steps taken to locate AP1 while the AGV is parked at stop 1?"* Here we define two atomic functions. $ap1step\text{-}stop1$ assigns a value of 1 for all states in which the manipulator increments an additional step in the spiral search localization pattern when attempting to locate the reflector AP1 while the AGV is parked at the first stop next to the RMMA, and 0 otherwise. The atomic function $ap1pass\text{-}stop1$ assigns a value of 1 for each state in which the manipulator detects the AP1 reflector while the AGV is parked at the first stop next to the RMMA, and 0 otherwise. The CTML formula for Query 4 is presented in Eqn. 4.

$$M\,ap1step\text{-}stop1\,U_{+}ap1pass\text{-}stop1 \tag{4}$$

For reasons similar to Query 1, and because the atomic function $ap1step\text{-}stop1$ does not have a match for the atomic proposition *true* in PCTL, this query could not be translated for PRISM.

4.2.5 Query 5. *"Given the AGV is parked at stop 1 and the manipulator fails to locate AP1, what is the probability that AP2 is located within 10 time units?"* We define an atomic function $ap2pass\text{-}stop1$ that assigns a value of 1 for each state in which the manipulator detects the AP2 reflector while the AGV is parked at the first stop next to the RMMA, and 0 otherwise. This query is similar to Query 3, except we must use a time bounded formula. First, we determine the probability to reach the states where the AP2 reflector is located within 10 time steps when the AGV is parked at the first stop next to the RMMA, starting from every possible state. Then, we filter out all but the

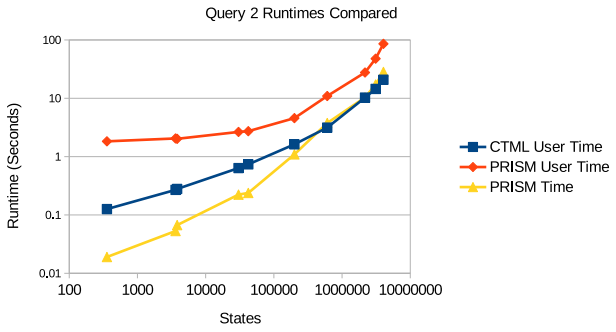

FIGURE 8. LINE PLOT COMPARING RUNTIMES OF QUERY 2.

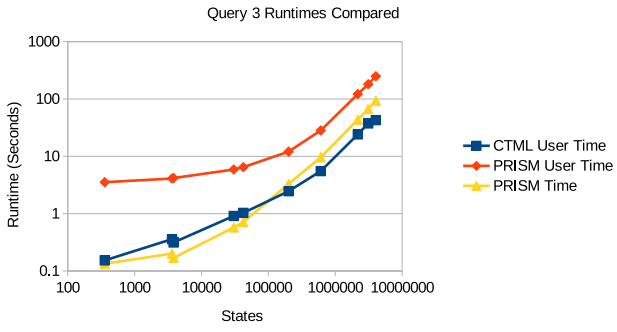

FIGURE 9. LINE PLOT COMPARING RUNTIMES OF QUERY 3.

states where the manipulator fails to locate AP1 while the AGV is parked at the first stop. We then sum, over all $ap1fail\text{-}stop1$ states, the probability to reach the first state multiplied by the probability to locate $ap2pass\text{-}stop1$ starting from the state. This quantity must be divided by $MoneU_{\times}\, ap1fail\text{-}stop1$, which has been explained in Query 2. The CTML formula for Query 5 is shown in Eqn. 5.

$$\frac{MoneU_{\times}((MoneU_{\times}^{\leq 10} ap2pass\text{-}stop1) * ap1fail\text{-}stop1)}{MoneU_{\times}\, ap1fail\text{-}stop1} \quad (5)$$

Query 5 could not be translated for use with the PRISM tool, since time-bounded operators are not expressible in PLTL [3,10].

4.2.6 Query 6.

"Given the AGV e-stops while the AGV is approaching stop 1 what is the expected time until the manipulator fails to locate AP1?" Similar to Queries 3 and 5, we first determine the expected time until the manipulator fails to locate the AP1 reflector when the AGV is parked at the first stop next to the RMMA, starting from each possible state. Then, we filter out all but the states where the AGV is e-stopped when attempting to reach the first stop next to the RMMA. We then sum over all $agvestop1$ states, the probability to reach the first state multiplied by the expected time to $ap1fail\text{-}stop1$ starting from that state. Finally, this quantity must be divided by $MoneU_{\times}\, agvestop1$. The CTML formula for Query 6 is presented in Eqn. 6.

$$\frac{MoneU_{\times}((MoneU_{+} ap1fail\text{-}stop1) * agvestop1)}{MoneU_{\times}\, agvestop1} \quad (6)$$

Query 6 could not be translated for use with PRISM because the tool does not support nested real-valued formulas [1].

5 RESULTS AND COMPARISON

The numerical results and CPU runtimes for each query evaluated using CTML and PRISM are listed in Tab. 1, which shows the relative precision of 10^{-4} for numerical results. Line plots comparing the runtimes between CTML and PRISM for Queries 2 and 3 are also provided in Fig. 8 and Fig. 9.

We now interpret the CTML results of the sample queries and compare the difference in query support between the two tools. The results for Query 1 yielded one time-step on average until the AGV e-stops as the AGV travels to the first stop next to the RMMA. When we compare the results of Queries 2 and 3, the probability of failing to detect AP1 is not influenced by the occurrence of an AGV e-stop since the result of both queries tend toward zero. It is also shown that, in absence of additional uncertainty, failing to detect the AP1 reflector is not a likely occurrence. This is further reinforced by the results of Query 4, which shows that between no steps and one step on average should be required to detect AP1 when the AGV is parked at the first stop. Query 5, however, shows that the probability of failing to detect AP1 at the first stop and then detecting AP2 within 10 times steps is 37%. The result for Query 6 shows that, given an AGV e-stop occurred, between 0 and 1 time-steps elapse on average until the manipulator fails to locate AP1. Out of the six queries evaluated using CTML, only two could be translated such that all of the generated models could be successfully and consistently evaluated using PRISM for the reasons given in Section 4.2. For Queries 2 and 3, the numerical results of PRISM were identical to those of CTML.

For the ease of implementation, preparing the model as input for PRISM was more complicated than CTML since the states, transitions, and labels of the model had to be split across three files for each generated DTMC model whereas CTML could define all needed structures in one file. Furthermore, CTML did not require separately defined reward and labeling functions, as was the case with PRISM. We were also unable to determine a way to explicitly define different, named reward functions for use with the property specification language. This restricted us to using only one explicitly-defined reward function at a time.

In terms of the runtime, the longest user time for executing a query in CTML corresponded with Query 1, which was evaluated

TABLE 1. SAMPLE QUERY RESULTS AND CPU RUN TIME

		Parameters ($E\ t\ r\ n\ N$)[2]									
		1, 1, 1, 1, 1	2, 2, 1, 2, 1,	4, 10, 1, 1, 1	4, 10, 1, 2, 1	3, 3, 1, 3, 1	4, 10, 1, 3, 1	4, 4, 1, 4, 1	4, 10, 1, 4, 2	4, 10, 1, 4, 3	4, 10, 1, 4, 4
		States									
		357	3,621	3,819	30,305	42,164	200,115	606,305	2,171,190	3,106,315	4,041,440
		Arcs (CTML)									
		421	4,352	4,702	37,384	51,139	247,054	739,504	2,683,829	3,840,954	4,998,079
		Arcs (PRISM)									
Query No.	**Tool**	501	5,189	5,637	44,789	61,055	295,909	883,669	3,212,934	4,597,559	5,982,184
Q1[3]	CTML	1.1118	1.0008	1.0011	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
User Time[4]		0.140	0.361	0.358	0.980	0.995	4.116	7.069	57.041	77.943	127.787
Q2a[5]	CTML	0.0049	0.0018	0.0000	0.0000	0.0006	0.0000	0.0002	0.0000	0.0000	0.0000
User Time		0.126	0.269	0.279	0.636	0.737	1.624	3.144	10.276	14.478	20.812
Q2b[5]	PRISM	0.0049	0.0018	0.0000	0.0000	0.0006	0.0000	0.0002	0.0000	0.0000	0.0000
User Time		1.829	2.050	2.020	2.652	2.729	4.568	10.935	27.677	47.722	85.786
PRISM Time[6]		0.019	0.053	0.067	0.223	0.238	1.094	3.761	10.350	17.231	28.285
Q3a[5]	CTML	0.0049	0.0018	0.0000	0.0000	0.0006	0.0000	0.0002	0.0000	0.0000	0.0000
User Time		0.153	0.359	0.314	0.911	1.031	2.461	5.521	24.174	37.526	42.835
Q3b[5]	PRISM	0.0049	0.0018	0.0000	0.0000	0.0006	0.0000	0.0002	0.0000	0.0000	0.0000
User Time		3.536	4.097	4.176	5.888	6.472	12.085	28.287	122.074	180.884	249.742
PRISM Time		0.134	0.201	0.169	0.575	0.711	3.3000	9.6280	43.7320	67.1710	93.4110
Q4[7]	CTML	0.0049	0.0091	0.0132	0.0132	0.0115	0.0132	0.0125	0.0132	0.0132	0.0132
User Time		0.143	0.321	0.298	0.758	1.012	2.471	6.122	27.254	34.247	45.738
Q5[5]	CTML	0.3333	0.3611	0.3704	0.3704	0.3704	0.3704	0.3704	0.3704	0.3704	0.3704
User Time		0.153	0.313	0.305	0.908	1.176	2.494	4.675	18.340	23.682	34.548
Q6[3]	CTML	0.0865	0.0386	0.0000	0.0000	0.0153	0.0000	0.0057	0.0000	0.0000	0.0000
User Time		0.163	0.370	0.387	1.011	1.097	4.730	8.783	41.010	55.517	68.300

[2] See Section 3.1; [3] Computed as time-steps; [4] User Mode CPU Time, in seconds, as measured by the `time` command line utility; [5] Computed as probability;
[6] Internally measured model-checking time, in seconds, offered by PRISM; [7] Computed as spiral search-steps.

in just over 2 minutes when run on a model with approximately 4 million states and almost 5 million arcs. When comparing the runtimes of Queries 2 and 3, it was found that the user time of CTML was consistently lower than the user time measured for PRISM as the number of states and arcs in the models increased. However, the internally measured PRISM model checking time, or "PRISM time", yielded the following. For Query 2, the PRISM time was slightly lower than the CTML user time for the models with less than 600,000 states. For models exceeding 600,000 states, the PRISM time exceeded the CTML user time. For Query 3, the CTML user time overtook the PRISM time after the model size exceeded 200,000 states.

6 CONCLUSION

In this work, we explored the practical utility of the theoretically established advantages of CTML by using the language to evaluate a set of sample queries on a modeled, manufacturing-based application. The benefits and practical application of CTML established in this paper allows for consideration of future extensions. For example, existing work has pursued the development of a formal verification framework for SysML activity diagrams in which activity diagrams were translated to Probabilistic Timed Automata (PTAs) for use with PRISM [17]. Alternative approaches opted to map the activity diagrams to DTMCs or Petri nets [18,19]. Integration of CTML into such frameworks could further enhance CTML ease of implementation by allowing the use of existing SysML models such as the previously developed mobile manipulator performance test model [8].

Furthermore, such work could facilitate an exploration towards integrating CTML with PLM software. Existing work has demonstrated interest in unifying MBSE with PLM with noted benefits including the ability of such models to adapt throughout the design process and to provide a centralized source of design reference [20]. Interest has also been shown in efforts to unify the use of temporal-logic based verification and MBSE within PLM tool chains [21,22]. In this work, Form-L language (described as being comparable to LTL) was integrated with the Modelica language to facilitate subsequent case study on requirements management for cyber-physical systems [21].

REFERENCES

[1] Jing, Y., and Miner, A. S., 2018. "Computation tree measurement language (CTML)". *Formal Aspects of Computing,* ***30***(3-4), August, pp. 443–462.

[2] Hansson, H., and Jonsson, B., 1994. "A logic for reasoning about time and reliability". *Formal Aspects of Computing,* ***6***(5), September, pp. 512–535.

[3] Costas, C., and Mihalis, Y., 1995. "The complexity of probabilistic verification". *Journal of the ACM,* ***42***(4), July, pp. 857–907.

[4] Kattepur, A., 2019. "Workflow composition and analysis in industry 4.0 warehouse automation". *IET Collaborative Intelligent Manufacturing,* ***1***(3), October, pp. 79–89.

[5] Jing, Y., and Miner, A. S., 2018. "Action and state based computation tree measurement language and algorithms". In 15th International Conference on Quantitative Evaluation of Systems, A. McIver and A. Horvath, eds., Vol. 11024 of *Lecture Notes in Computer Science*, Springer Cham, pp. 190–206.

[6] Bostelman, R. V., Hong, T., and Marvel, J. A., 2015. "Performance measurement of mobile manipulators". In Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications, J. J. Braun, ed., Vol. 9498, International Society for Optics and Photonics, SPIE, pp. 97 – 106.

[7] Bostelman, R. V., Hong, T., and Marvel, J., 2016. "Survey of research for performance measurement of mobile manipulators". *Journal of Research (NIST JRES),* ***121***, June, pp. 342–366.

[8] Bostelman, R. V., Foufou, S., Hong, T., and Shah, M., 2017. "Model of mobile manipulator performance measurement using sysml". *Journal of Intelligent and Robotic Systems,* ***92***, September, pp. 65–83.

[9] Zurawski, R., and Zhou, M., 1994. "Petri nets and industrial applications: A tutorial". *IEEE: Transactions on Industrial Electronics,* ***41***(6), December, pp. 567–583.

[10] Kwiatkowska, M., Norman, G., and Parker, D., 2011. "PRISM 4.0: Verification of probabilistic real-time systems". In 23rd International Conference on Computer Aided Verification, G. Gopalakrishnan and S. Qadeer, eds., Vol. 6806 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 585–591.

[11] Bostelman, R. V., Foufou, S., Legowik, S. A., and Hong, T. H., 2016. "Mobile manipulator performance measurement towards manufacturing assembly tasks". In Product Lifecycle Management for Digital Transformation of Industries - 13th IFIP WG 5.1 International Conference, PLM 2016, Revised Selected Papers, B. Eynard, A. Bouras, A. Bernard, L. Rivest, and R. Harik, eds., Vol. 492 of *IFIP Advances in Information and Communication Technology*, Springer New York LLC, pp. 411–420.

[12] Bostelman, R. V., Eastman, R., Hong, T. H., Aboul-Enein, O., Legowik, S. A., and Foufou, S., 2016. "Comparison of registration methods for mobile manipulators". In Advances in Cooperative Robots, pp. 205–213.

[13] Ciardo, G., and Miner, A. S., 2004. "Smart: the stochastic model checking analyzer for reliability and timing". In 1st International Conference on Quantitative Evaluation of Systems, IEEE, pp. 338–339.

[14] Parker, D., Norman, G., and Kwiatkowska, M., 2019. Prism manual. On the WWW, April. URL `http://www.prismmodelchecker.org/manual/Main/AllOnOnePage`.

[15] Parker, D., Norman, G., and Kwiatkowska, M., 2020. Prism changelog. On the WWW, April. URL `https://github.com/prismmodelchecker/prism/blob/master/CHANGELOG.txt`.

[16] Kwiatkowska, M., 2007. "Quantitative verification: models techniques and tools". In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ESEC-FSE '07, Association for Computing Machinery, pp. 449–458.

[17] Baouya, A., Bennouar, D., Mohamed, O. A., and Ouchani, S., 2015. "A quantitative verification framework of sysml activity diagrams under time constraints". *Expert Systems with Applications,* ***42***(21), November, pp. 7493–7510.

[18] Jarraya, Y., Soeanu, A., and Debbabi, M., 2007. "Automatic verification and performance analysis of time-constrained sysml activity diagrams". In 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE, pp. 515–522.

[19] Huang, E., McGinnis, L. F., and Micthell, S. W., 2019. "Verifying sysml activity diagrams using formal transformation to petri nets". *Systems Engineering,* ***23***(1), November, pp. 118–135.

[20] Bajaj, M., Zwemer, D., Yntema, R., Phung, A., Kumar, A., Dwivedi, A., and Waikar, M., 2016. "Mbse++ — foundations for extended model-based systems engineering across system lifecycle". *INCOSE International Symposium,* ***26***(1), September, pp. 2429–2445.

[21] Garro, A., Tundis, A., Bouskela, D., Jardin, A., Thuy, N., Otter, M., Buffoni, L., Fritzson, P., Sjölund, M., Schamai, W., and Olsson, H., 2016. "On formal cyber physical system properties modeling: A new temporal logic language and a modelica-based solution". In 2016 IEEE International Symposium on Systems Engineering, IEEE, pp. 1–8.

[22] Aiello, F., Garro, A., Lemmens, Y., and Dutré, S., 2017. "Simulation-based verification of system requirements: An integrated solution". In 2017 IEEE 14th International Conference on Networking, Sensing and Control, pp. 726–731.