Improving Verification Accuracy of CPS by Modeling and Calibrating Interaction Uncertainty

WENHUA YANG and CHANG XU, State Key Laboratory for Novel Software Technology and Department of Computer Science and Technology, Nanjing University
MINXUE PAN, State Key Laboratory for Novel Software Technology and Software Institute, Nanjing University
XIAOXING MA and JIAN LU, State Key Laboratory for Novel Software Technology and Department of Computer Science and Technology, Nanjing University

Cyber-Physical Systems (CPS) intrinsically combine hardware and physical systems with software and network, which are together creating complex and correlated interactions. CPS applications often experience uncertainty in interacting with environment through unreliable sensors. They can be faulty and exhibit runtime errors if developers have not considered environmental interaction uncertainty adequately. Existing work in verifying CPS applications ignores interaction uncertainty and thus may overlook uncertainty-related faults. To improve verification accuracy, in this article we propose a novel approach to verifying CPS applications with explicit modeling of uncertainty arisen in the interaction between them and the environment. Our approach builds an Interactive State Machine (ISM) network for a CPS application and models interaction uncertainty by error ranges and distributions. Then it encodes both the application and uncertainty models to SMT formula to leverage SMT solvers searching for counterexamples that represent application failures. The precision of uncertainty model can affect the verification results. However, it may be difficult to model interaction uncertainty precisely enough at the beginning, because of the uncontrollable noise of sensors and insufficient data sample size. To further improve the accuracy of the verification results, we propose an approach to identifying and calibrating imprecise uncertainty models. We exploit the inconsistency between the counterexamples' estimate and actual occurrence probabilities to identify possible imprecision in uncertainty models, and the calibration of imprecise models is to minimize the inconsistency, which is reduced to a Search-Based Software Engineering (SBSE) problem. We experimentally evaluated our verification and calibration approaches with real-world CPS applications, and the experimental results confirmed their effectiveness and efficiency.

CCS Concepts: • **Software and its engineering → Software verification and validation**; *Search-based software engineering*; • **Computer systems organization → Embedded and cyber-physical systems**;

Additional Key Words and Phrases: Verification, cyber-physical systems, calibration

This work is supported in part by National Basic Research 973 Program (Grant #2015CB352202), National Natural Science Foundation (Grant #61472174, #61690204), and Natural Science Foundation of Jiangsu Province (Grant #BK20150589) of China.

Author's addresses: W. Yang, C. Xu, X. Ma and J. Lu, Department of Computer Science and Technology, Nanjing University; emails: ywh.nju@outlook.com, {changxu, xxm, lj}@nju.edu.cn; M. Pan, Software Institute, Nanjing University; email: mxp@nju.edu.cn. Corresponding author: C. Xu.

1533-5399/2017/9-ART39 $15.00
https://doi.org/0000001.0000001

ACM Reference format:
Wenhua Yang, Chang Xu, Minxue Pan, Xiaoxing Ma, and Jian Lu. 2017. Improving Verification Accuracy of CPS by Modeling and Calibrating Interaction Uncertainty. *ACM Trans. Internet Technol.* 9, 4, Article 39 (September 2017), 37 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

Internetware is envisioned as a new software paradigm for resource integration and sharing in the open, dynamic and autonomous network environment. This new paradigm emphasizes the autonomy of the constituent services of a system, the flexibility of coordination among the services, as well as the adaptability, evolvability and trustworthiness of the system [32]. As computing devices, human society, and physical objects are becoming seamlessly integrated together, Internetware must be ready to support the software systems that will orchestrate information, processes, decisions, and interactions in this Internet environment [35]. Typical examples of Internetware systems include Internet-based and Cyber-Physical Systems (CPS). These systems are often characterized with autonomy, coordination, context-awareness and self-adaptability. Software development for such kind of CPS is a typical target domain of the Internetware paradigm because the constituting entities are autonomous, the environment is open and constantly changing, the system is built with the coordination of these autonomous entities and needs to automatically adapt to the changing environment [31].

CPS are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core [30]. They have wide applicability in various economically vital domains including high confidence medical devices and systems, advanced automotive systems, traffic control and safety, energy, defense, aerospace and distributed robotics, etc. [30, 39]. CPS typically consist of physical and cyber spaces, with various sensors and actuators. These spaces integrate computation and physical processes, and are interconnected by a network layer for exchange of data. Usually, the computation and network monitor the physical processes with sensors and make controls based on application logics, with feedback loops where physical processes affect computations.

1.1 Motivation

CPS intrinsically combine hardware and physical systems with software and network, which are together creating complex and correlated interactions. Uncertainty is prevalent in CPS, either in the software platform [14], the middleware layers [15], or in the interactions between the software and physical systems [6]. This brings challenges to build dependable CPS, since developers have to guarantee the correctness of cyber systems' predefined business logics while considering the various uncertainty. Real-world CPS are thus error-prone [1], which makes their quality assurance vitally important and significant.

Testing is an essential activity in engineering and is widely used in industry to guarantee the quality of any type of system. To assure the quality of CPS, many research efforts focus on testing techniques. Some techniques [3, 47, 57] use model-based testing to generate test cases for CPS, and some work uses runtime monitor to dynamically detect faults in CPS [26]. Simulation is often combined with field testing to test CPS [24, 26, 57], since it can reduce testing cost. Multiple parts of CPS are often embedded systems with a real-time software executing. Therefore, there are some pieces of work focusing on testing functional or non-functional correctness (e.g., temperature, power consumption, and timing) of real-time CPS [21, 24, 50]. Testing CPS poses a big challenge, since it may contain hardware and software testing, computation and communicational testing,

functional and non-functional testing for each cyber and physical component individually, and system testing to test the complete system. It is generally infeasible to predict and enumerate all possible conditions that a CPS application can encounter at runtime. Therefore, issues concerning testing loom large.

On the other hand, formal verification, which is regarded as an effective method to achieve dependability, can provide evidence that the set of stated functional and non-functional properties are satisfied during the system's operation [5]. Recent studies have also reported promising results in applying these techniques to verify properties of CPS, e.g., safety [18, 41], robustness [49] and performance [28]. Technically, researchers use model checking [2, 11] or theorem proving [33, 44] to verify CPS. However, when it comes to verifying CPS under real-world environments, much existing work lacks of considering CPS's interaction uncertainty with physical environment.

CPS typically comprise of physical and cyber spaces. In the physical space, they often interact with physical environment in a feedback way. First, they sense the environment and react to the environmental changes by actuators, and the action upon certain environmental changes can then have an effect on the environment. CPS are gaining increasing attention for their ability to bridge the gap between the cyber and physical worlds, however, assuring the quality of CPS is challenging because interactions between physical and cyber components are complex and often unpredictable [59]. For example, an unforeseen interaction among the control modules and the physical environment, led to a safety violation and an unfortunate disqualification of a DARPA Urban Challenge vehicle, named "Alice" [36]. A CPS application's environmental interaction with the physical world can be affected by uncertainty, which is inevitably caused by unreliable environmental sensing [6, 56]. For environmental sensing, an application may only be able to obtain an estimate of its environmental conditions, but never knows its real state. Consider an example robot-cars application [55, 56] that aims to explore an unknown area without bumping into any obstacle. The car can sense its four-directional distances (front, back, left and right) to nearby obstacles using its built-in ultrasonic sensors. Based on these sensed distances, it decides its next movement to avoid obstacles. However, the sensed distances may not faithfully reflect real distances from the car to its nearby obstacles, since the sensing always contains unpredictable noise. Such sensing uncertainty can cause inconsistency between an application's understanding to its environment and its actual environmental conditions, thus affecting the application's functionalities. Overlooking such uncertainty can lead to inaccurate verification results. However, such uncertainty has not been well studied for the verification problem of CPS applications.

1.2 Research Work

In this article, to improve verification accuracy, we propose a novel approach to verifying CPS applications with explicitly modeling the applications' environmental interaction uncertainty with the physical world. An SMT solver is leveraged to verify a CPS application model with uncertainty modeled by approximated error ranges and distributions. Specifically, we build an ISM network for a CPS application and collect the application's behavior by exploring each potential path within a bound of path length to get a path condition, in which environment-related variables are augmented with error ranges from the uncertainty model. The application's failures are depicted by assertions (failure conditions), which are joined with the path conditions and checked for satisfiability. If satisfied, one concrete solution will be returned by the solver [12]. The path and the corresponding solution are comprised together as a *counterexample*, which corresponds to an application failure behavior. We also devise a means to calculate the counterexamples' probabilities to help developers focus on more likely faults.

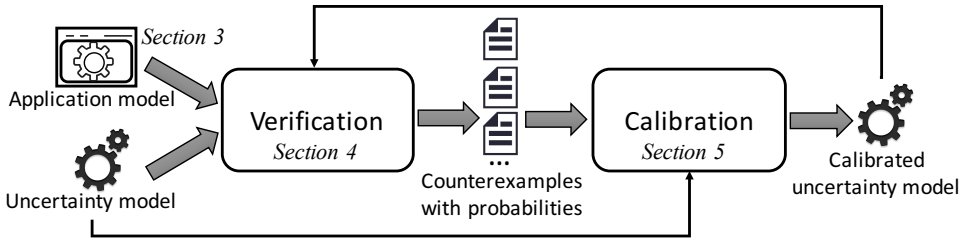


Fig. 1. Overview of verification and calibration.

One key input in the verification process is the uncertainty model, as illustrated in Figure 1. Error range and distribution are commonly used in physics to specify uncertainty. Our work introduces this practice to model environment-interacting uncertainty. However, it may be difficult to specify the uncertainty arisen in the interaction between system and environment precisely and completely because of the uncontrollable noise of sensors employed for monitoring and insufficient data sample size. As a result, such verification results can deviate from reality when the uncertainty specification used in verification is imprecise: less important counterexamples can be falsely highlighted, or even worse, false counterexamples may be reported. To eliminate imprecise uncertainty model's effect on verification results, we propose an approach adopting the idea of Search-Based Software Engineering (SBSE) to calibrating the uncertainty model. The calibration problem is represented as an SBSE problem, whose goal is to minimize the differences between the counterexamples' calculated probabilities and actual probabilities. The calibrated uncertainty model can improve the accuracy of the verification results. The novelty of the calibration approach is in its application of SBSE for the particular problem of uncertainty model calibration for CPS applications. There were no previous works in this direction.

In an evaluation on real-world CPS applications, our verification approach reported 1.6-9.0x more real counterexamples than the approach not considering interaction uncertainty, which demonstrates that our approach has an improved accuracy. Meanwhile, the experimental results also showed that our calibration approach improved the precision of imprecise uncertainty models approximately to the precise level. We summarize our contributions in this article below:

- We propose to explicitly consider uncertainty in verifying CPS applications, to improve verification accuracy, and to rank counterexamples according to their probabilities, to help developers focus on more likely faults.
- We propose to calibrate imprecise uncertainty model with a search-based approach exploiting the inconsistency between the counterexamples' calculated probabilities and actual probabilities, to further refine verification results.
- We evaluate our verification and calibration approaches with CPS applications by both real and simulation experiments, and validate their usefulness in practice.

The remainder of this article is organized as follows. Section 2 uses a motivating example to explain the inadequacy of existing work and motivate our work. Section 3 introduces our modeling of CPS applications. Section 4 presents our verification approach in detail. Section 5 presents our calibration approach in detail. Section 6 evaluates our verification and calibration approaches with CPS applications. Section 7 discusses related work, and finally Section 8 concludes this article.

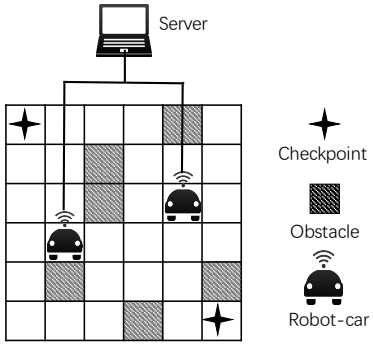


Fig. 2. Example robot-cars application.

2 MOTIVATING EXAMPLE

In this section, we present a CPS example to motivate our work and explain the problem. As illustrated in Figure 2, the example consists of two robot-cars that collectively explore an unknown area and a server that monitors the cars' states such as battery level and travel distance, etc. In the area, there are many obstacles that the cars must avoid, and several locations marked as checkpoints that the cars must pass. Each car can sense its four-directional distances (front, back, left and right) to nearby obstacles using its built-in ultrasonic sensors. Based on these sensed distances, it decides its next movement to explore the area and avoid obstacles. A failure occurs when any car bumps into an obstacle. Each car is equipped with an RFID reader and each checkpoint has an RFID tag, so the car would receive a signal when it passes a checkpoint.

There is a server monitoring the cars' states. They are connected by WiFi in a local area network, and together constitute the example CPS application. The server has many responsibilities, such as checking whether the battery level is too low or any car has bumped into an obstacle. Among them, the major responsibility is to check whether all checkpoints have been discovered within some certain travel distance. We consider it a failure if the cars have traveled a distance exceeding a given bound but not discovered all checkpoints, since it suggests a problem in the cars' logic design for the area. To motivate our work, we present a simple but representative design of the cars' controlling logic. In this design, the car moves in a zig-zag style, while avoiding to bump into any obstacles. Specifically, before each move, the car first senses its four-directional distances. Based on the sensing distances, the car performs an action following the associated rules to its current state. Different states have different rules, which assures that the car makes the correct turns to form the zig-zag routines. Due to page limit, we just present the rules associated with the car's initial state, but this suffices for explaining our problem. There are four rules determining the car's action:

- Rule 0. If the front distance is greater than or equal to 15cm, then move forward for 10cm;
- Rule 1. If the front distance is less than 15cm and the left distance is greater than or equal to 15cm, then turn left, move forward for 10cm, and turn left again;
- Rule 2. If the front distance and the left distance are both less than 15cm and the right distance is greater than or equal to 15cm, then turn right, move forward for 10cm, and turn right again;
- Rule 3. If the front, left and right distance are all less than 15cm, then move back for 10cm.

As in a CPS application, multiple subsystems need to collaborate with each other to function properly, so we also need to provide the rules for the server. For simplicity, we only give one rule which specifies the major responsibility that checks whether all checkpoints have been discovered

within some certain travel distance. In this example, there are two checkpoints, so the server uses two boolean variables c_1 and c_2 to track whether the checkpoints have been discovered. The server also records the total distance that the two cars have traveled so far; if the distance exceeds 10000cm, it aborts the area exploration and reports a failure. Therefore, we can have the rule for the server as:

- If c_1 is false or c_2 is false, and the two robot-cars' total travel distance is greater than or equal to 10000cm, then reports a failure.

Suppose one wants to use model checking to verify this application. Typically, a model of the application is first built, and various model checking techniques can be applied to check the correctness. However, real counterexamples (i.e., true positives) may also be missed if one does not consider uncertainty in verifying this application. Consider one of the failure conditions that any car bumps into an obstacle. Since the rules require that the front distance be at least 15cm for the car to move forward and each time the car moves forward only for 10cm, one may conclude that the car will never bump into an obstacle in the front. But in reality, environmental sensing can contain uncertainty as explained earlier. Suppose that sensing distance is subject to an error range of $[-6, 6]$, and the front distance for the car is 9cm. Then due to uncertainty, the sensed front distance could be 15cm, so the car would move forward and bump into an obstacle in this situation. Having not considered uncertainty, traditional verification methods cannot find such uncertainty-related faults. Therefore, this calls for new effort to model such uncertainty so that one can discover such potential problems in a CPS application in a more accurate way. Our approach explicitly models interaction uncertainty and combines it with the application model for verification, so the counterexample it revealed would contain not only the sensed environmental data but also the actual environmental data and the uncertainty, such as a sensing error. For example, for the aforementioned failure, our approach is able to report a counterexample suggesting that when the sensed front distance is 15cm, the sensing error is −6cm and the actual front distance is 9cm, the car is moving forward and will bump into an obstacle.

Modeling such interaction uncertainty is vital for the verification. However, it may be difficult to specify the uncertainty precisely enough at the beginning, because of the uncontrollable noise of sensors and insufficient data sample size. Take the robot-cars application for example again. The error range for the sensing distance should be $[-6, 6]$, but one may obtain an imprecise uncertainty model of $[-4, 4]$. Then for the same scenario that the front distance for the car is 9cm, based on this imprecise uncertainty model, a conclusion that the car would not move forward and bump into obstacles can be drawn. Obviously, it misses the aforementioned failure. To further improve the accuracy of verification results, it is essential to have a precise uncertainty model. Our approach can effectively identify and calibrate possible imprecise uncertainty models, and thus obtain more accurate counterexamples.

3 MODELING CPS APPLICATIONS

Unlike traditional embedded systems, a CPS application is typically designed as a network of interacting elements with physical inputs and outputs instead of as standalone devices [30]. The rise in popularity of hardware devices has increased interest in the area of cyber-physical systems. Multiple sensory input or output devices, such as GPS chips and proximity sensors can be heavily used in cyber-physical systems, as they are essential for cyber-physical systems to interact with the physical world. Many applications of CPS fall under sensor-based communication-enabled autonomous systems. For example, some CPS applications exploit the physical information collected by sensor networks to bridge the physical and cyber spaces and relay the information to a central node [54]. More open infrastructures such as cloud, can also be employed to store and process

information in CPS applications. Nevertheless, the interaction with physical world is still mostly conducted by sensory devices. CPS typically interact with physical environment in a feedback way. First, the application senses its environment to capture interesting environmental changes. Then, according to its predefined logics, the application makes a decision by selecting appropriate action to such environmental changes. The action can change the environment and affect the application's environmental sensing afterwards. The sensing can be uncertain, which is caused by imperfect sensing technologies nowadays. Since there is no existing modeling approach for CPS application that explicitly consider an application's interaction effects on its environmental sensing afterwards and the uncertainty occurring during the interaction, we propose a new model, named Interactive State Machine (ISM) to meet this requirement.

The ISM is defined as a tuple $M := (S, V, R, s_0)$, in which symbols are explained below:

- S is a set of this application's all states, and $s_0 \in S$ is its initial state, with which the application starts.
- V is a set containing this application's all variables. $V = V_s \cup V_n$, in which V_s and V_n represent two disjointed categories. V_s contains all *sensing variables*, which store values of environmental attributes interesting to this application (updated by relevant sensing devices). V_n contains other normal variables, i.e., *non-sensing variables*.
- R is a set containing this application's all rules. For each rule $r \in R$, r is associated with a state $s \in S$, which is r's source state. Rule r takes a form of $r := (condition, actions)$. $condition$ is a logical formula built on V, and its satisfaction would trigger the execution of this rule. $actions$ specifies what should be done when executing this rule. $actions$ can include $internal\ actions$ and $interactive\ actions$. The former takes internal actions by updating values of non-sensing variables, e.g., updating the application's current state, and in this case the application transits to a new state. The latter takes interactive actions by interacting with the application's environment directly, e.g., making a robot-car move forward. $interactive\ actions$ can also specify the extents of their effects on the environment by updating certain non-sensing variables, e.g., recording the distance that the car has moved forward.

ISM is executable. Starting from its initial state s_0, an ISM $M := (S, V, R, s_0)$ repeatedly reads values of its sensing variables (automatically updated by environmental sensing), then evaluates and decides which rule to execute, and finally conducts the executed rule's associated actions. When a state $s \in S$ is set as M's current state, rules having this state as source state are enabled, while other rules are disabled. Only enabled rules participate in rule evaluation upon each environmental sensing. When an enabled rule r's condition $r.condition$ is satisfied, the rule is triggered for execution. If multiple rules are triggered, only one of them is selected to execute. This tie can be resolved by some priority or random mechanisms, which are not our focus in this article and therefore omitted. When a rule r is selected to execute, by default its actions $r.actions$ are conducted in a sequential way. Conducting actions concurrently is possible by allowing that all the concurrent actions in one rule to be executed on variable values of the same version, and this needs to be specified beforehand. Thus, an ISM M's execution can be conceptually modeled by a path which is a sequence of states and rules: $\sigma = s_0\ r_1\ s_1 \ldots r_n\ s_n$. Similar to traditional programming, we define $path\ condition$ of execution σ as:

$$pc(\sigma) = \bigwedge_{i=1}^{n} r_i.condition$$

To be representative, we adopt a quantifier-free first-order logic based language for specifying a rule's condition. With this language, a rule's condition can be specified by a logical formula that is

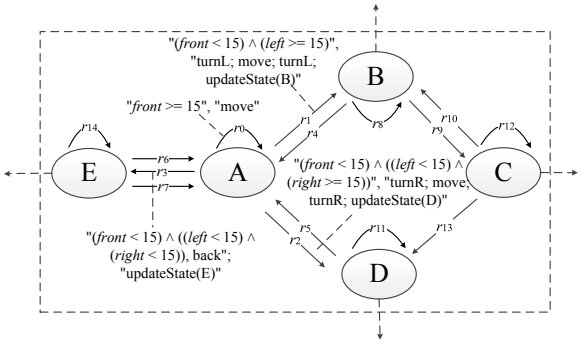


Fig. 3. Partial ISM model of the robot-cars application.

recursively constructed using the following syntax:

$$f := (f) \text{ and } (f) \mid (f) \text{ or } (f) \mid (f) \text{ implies } (f) \mid \text{not } (f) \mid \text{bfunc}(v, \ldots, v).$$

Note that the above "and", "or", "implies" and "not" logical connectives follow their traditional interpretations, i.e., representing conjunction, disjunction, implication and negation operations, respectively. Terminal *bfunc* refers to any user-defined or domain-specific function that returns either true or false. When a *bfunc* is easy to understand, one may also use its corresponding operator to simplify its representation, e.g., "$largerThan(v, 50)$" can also be represented by "$v > 50$".

Take the robot-cars application for example, we can build an ISM for one of the cars. We illustrate in Figure 3 part of the ISM, which starts from the initial state in consistent with the presentation in last section. The ISM model is: $M := (S, V, R, s_0)$, where $S := \{A, B, \ldots, E, \ldots\}$, $R := \{r_0, r_1, \ldots, r_{14}, \ldots\}$, V is the set of variables used in this application (in particular, by R), and the application's initial state s_0:= A. As mentioned in last section, there are four rules associated with State A, i.e., having A as their source states: r_0, r_1, r_2 and r_3 (multiple actions are sequentially separated by semicolon ";"):

r_0 := ("$front \geq 15$", "move").
r_1 := ("$(front < 15) \wedge (left \geq 15)$", "turnL; move;
turnL; updateState(B)").
r_2 := ("$(front < 15) \wedge ((left < 15) \wedge (right \geq 15))$",
"turnR; move; turnR; updateState(D)").
r_3 := ("$(front < 15) \wedge ((left < 15) \wedge (right < 15))$",
"back; updateState(E)").

These rules are the formal forms of the four rules presented in the last section. They refer four variables: $front$, $back$, $left$ and $right$, which are all sensing variables, representing sensed distances between the car and its four-directional obstacles (front, back, left and right), respectively. If one rule's condition is satisfied, it may invoke some actions. For example, they can be interactive actions move and back, which mean driving the car to move forward and backward by a unit distance (say, 10cm), respectively. Interactive actions turnL and turnR represent turning the car left and right by 90°, respectively. After these interactive actions, the application can also conduct a state-transition

internal action, e.g., r_0's $updateState(B)$, which transits the application from state A to a new state B. At State B, new rules associated with this state (r_4, r_8, r_9) are enabled while the previous rules (r_0, r_1, r_2,r_3) are all disabled.

ISM network. Common applications of CPS typically consist of multiple autonomous or semi-autonomous components that collaborate with each other to fulfill some certain tasks. Capturing large and complex systems with a single model is not practical. Instead, such systems are typically built by combining a number of simpler components. It is natural to model each component separately, and then compose the components [9]. Each component has its own controlling logic, and can run concurrently and communicate with other components. To adequately specify this feature, we propose the ISM network to model a CPS application. The ISM network is composed of multiple ISMs, each of which specifies the controlling logic of one autonomous or semi-autonomous component. The execution of the ISM network is conducted by executing all the ISMs concurrently, which corresponds to the concurrent running of components in a CPS application.

Furthermore, we need to model the communication and collaboration among components. In a CPS application, different components need to communicate to share information. The communication can be performed through many forms, such as LAN and WiFi communication. Since our focus is the correctness of controlling logics on the CPS application level, we assume that the communication has been well designed to be reliable, and we only need to specify the communication actions and their effects on the applications, which are the information transmitted in the communication process and observed by the receiving components. In ISMs, information is modeled by variables. Communication is achieved via shared variables [22]. This is natural when modeling concurrent processes or threads [9]. Specifically, when one component is sending information, we model this action by allowing the corresponding receiving ISM to update its concerned shared variables. The update can be observed by other ISMs that also have these shared variables, and this models the information receiving process. Besides the ability to specify communications among components in a CPS application, we also support temporal coordination of actions, since there are many scenarios where some actions from different components running concurrently need to be performed simultaneously. To achieve this, we introduce synchronization labels [22] to its rules, and require the rules with the same labels to be executed at the same time.

Instead of combining the paths of all the ISMs with careful synchronization, a more general approach to obtaining the execution of the ISM network is to parallel compose all the ISMs into a new ISM and then extract paths from the new ISM, similar to the parallel composition of timed automata [46]. The composition is conducted in an iterative way: first composing two ISMs to form a new ISM, then composing the new ISM and another ISM, until there is only one ISM left. There is a special case of parallel composition called *synchronous composition* that require all the ISMs must fire one rule at each step. In synchronous composition, all rules can be regarded to possess the same synchronous label. In our robot-cars application, the two cars share the same logic, and therefore their ISMs are identical. For the two cars' coordination, we require the two cars to move in the same pace, i.e., the two ISMs would fire each of their own rules at the same time. This can be achieved by synchronous composition of the two ISMs, which results in that in the composed product of the two cars' ISMs, every rule is composed of two rules, each of which comes from one car's ISM, so that the two cars make actions in the same pace.

There is one more thing to be noted. In the robot-cars application, the server does not make actions but only monitors the cars' states by receiving data from the cars, i.e., it has no its own states but only evaluates on the cars' states. Therefore, the server's function is not necessary to be modeled by an ISM. It can be abstracted as constraints on the cars' states. Since these constraints

are about the properties to be verified, we postpone the description to the next section when failure conditions are formulated.

4 VERIFYING CPS APPLICATIONS

In this section we present our approach to verifying CPS applications with interaction uncertainty. For a CPS application, we build ISM models for each component, and compose them to model the entire application. We also need to specify the environment, as a CPS application can demonstrate different behaviors in diverse environments. Take the motivating robot-cars application for example, the cars could be placed in different initial positions with different directions of any area where obstacles are put with high disparities, and consequently take different moving routes. Our aim is to verify whether a CPS application will fail in a specific environment. A failure is defined as a violation of failure conditions (assertions) which are different application by application. As discussed in the motivating example shown in Figure 2, the failure condition of our robot-cars application is that any car bumps into obstacles or the cars have traveled a distance exceeding a given bound but not discovered all checkpoints. Suppose the bound of traveled distance d is 10000cm, and two boolean variables c_1 and c_2 are used to specify whether checkpoint 1 and 2 have been visited, respectively. Then the failure condition can be formulated as $front^1 \leq 0 \vee front^2 \leq 0 \vee ((c_1 == false \vee c_2 == false) \wedge d \geq 10000)$, where $front^1$ and $front^2$ are the front distances of Car A and B, respectively.

The overview of our verification approach is shown in Algorithm 1. The algorithm takes input a set of ISMs LM, an uncertainty model U, an environment model E, and a failure condition fc as the property to be checked. Since we adopt the bounded model checking approach, a bound k is also required as an input. First, the algorithm composes the ISMs in LM to get a new model M. The main data structure used in the algorithm is a list $path$, which records the path that is currently being explored. The algorithm traverses M in a depth-first manner to find new paths (Lines 5-9, Lines 19-20). Since the number of optional paths could be infinitely many, for practical considerations, our approach bounds the length of a path being explored with a configurable integer during the traversal (Line 6: $i < k$). The traversal ends when there is no unexplored path within the bound (Line 22).

For each selected path, the algorithm extracts its path condition pc. The ISM explicitly specifies the reacting logics of an application, but does not include its uncertainty and environmental settings. So we modify the path condition by introducing uncertainty, and augment the path and failure condition with environmental settings for realistic verification. Then the concatenation of path and failure condition is passed to Z3 [12], an efficient SMT solver, to check whether it can be satisfied. If so, a counterexample is found. It is possible that many counterexamples can be reported. To make the verification results more actionable to users, our approach prioritizes the reported counterexamples according to their occurrence probabilities. In the following three subsections, we present our ideas of dealing with uncertainty, modeling environments, and prioritizing counterexamples in detail.

4.1 Dealing with Uncertainty

CPS often consist of heterogeneous physical units (e.g., sensors, control modules) communicating via various networking equipment, interacting with physical world, applications and humans. Thus, uncertainty is intrinsic in CPS due to novel interactions of embedded systems, physical environment, networking equipment, cloud infrastructures and humans [3, 58]. It comes in different locations, in various forms and can be measured in different ways, e.g., the execution time of a component in a CPS application can be uncertain. One major source of uncertainty is an applications' environmental interaction with its physical world, which is naturally caused by unreliable sensing [6, 56]. In this

ALGORITHM 1: Verification algorithm

Input: Set of ISMs LM, uncertainty model U, environment model E, failure condition fc, and bound k.
Output: Set C of counterexamples with probabilities.

```
1: C := ∅;
2: Compose LM to a new ISM M := (S, R, V, s0);
3: path :=< s0 >; i := 0;
4: repeat
5:     s := the last state of path;
6:     if i < k && s has unexplored rules then
7:         r := an unexplored rule of s;
8:         s' := the state that r leads to;
9:         append r and s' to path; i := i + 1;
10:        extract the path's path condition pc;
11:        modify pc by introducing uncertainty from U;
12:        augment pc and fc with environmental settings from E;
13:        check pc && fc with a constraint solver;
14:        if pc && fc is satisfied then
15:            estimate the probability of the counterexample;
16:            add the counterexample with probability to C;
17:        end if
18:    else
19:        remove s' and r from path;
20:        i := i − 1;
21:    end if
22: until path == ∅;
23: return C;
```

article, we pay special attention to this environmental interaction uncertainty, as it can cause inconsistent understandings to an application between its sensed environment and its faced actual environment, and the application may fall into failure due to such imprecise understanding to the its environment. Thus, we should consider environmental interaction uncertainty in verifying CPS applications. Otherwise, the accuracy of verification results cannot be guaranteed. To properly specify environmental interaction uncertainty caused by unreliable sensing for the purpose of verification, we studied the related approach in other disciplines such as mechanics [52] and some real cases from our past work [56]. We observed that environmental interaction uncertainty demonstrates regular patterns. The sensed value of unreliable sensing often falls into an error range, with a distribution determined by the physical characteristics of sensing technologies. We call this type of uncertainty the *environmental interaction uncertainty*, and in this section, we explain how to model this uncertainty in the verification of CPS applications.

Uncertainty affects CPS applications' understanding to environment, and thus affects values of sensing variables that represent the sensed environments. Given an application's ISM, to model uncertainty, we first need to identify the variables in the path condition that would be affected by uncertainty. Then for each of these variables, we give an error range and a distribution for its potential value, i.e., for a variable v affected by uncertainty, let $[a, b]$ $(a < b)$ be the error range of v. Then the lower bound and upper bound of variable v are $v + a$ and $v + b$ respectively. Meanwhile, we set a distribution p of the variable's value between its lower and upper bounds. The lower bound, upper bound and the distribution of a variable can be obtained from field studies or experiments

with statistical analysis. In the robot-cars application, the distance $front$ between the car and its front obstacle is affected by uncertainty. We found that the ultrasonic sensor used in the robot-cars has an error in sensing. Field studies show that its error range is $[-6, 6]$ cm and its error distribution is a Gaussian one. So, for variable $front$, we give an error range of $[-6, 6]$. Then the lower bound and upper bound are $front-6$ and $front+6$, respectively. The distribution of $front$'s value between its lower and upper bounds is a Gaussian distribution.

Now we can show how to augment a path condition with uncertainty. In a path condition, for all its variables, there is no uncertainty considered. Therefore, for each variable v affected by uncertainty, since v does not include uncertainty, we use a new variable v' to represent v with uncertainty. v' satisfies the constraint $v + a \leq v' \leq v + b$, where $[a, b]$ is the error range of v. This means that the value of v' can range from $v + a$ to $v + b$. Clearly, for environmental sensing, the value of v' is a sensed value of the application, and the value of v is the actual value about the environment. We also set a distribution $P(v')$ for v' that will be used to prioritize reported counterexamples, which will be explained later. Then we replace v with v' in the path condition, and join the constraint $v + a \leq v' \leq v + b$ with the condition, which forms a new path condition.

Consider the robot-cars application example. For the ISM composed by the two cars' ISMs M^1 and M^2, let's take a short path $\sigma = (A^1A^2)\ r\ (A^1A^2)$ for illustration. Here, (A^1A^2) is the composing state of A^1 and A^2 which correspond to the state A in M^1 and M^2, respectively. Rule r is the composition of rules r_0^1 and r_0^2 which correspond to the rule r_0 in M^1 and M^2, respectively. Since there is only one rule in σ, the path condition of σ is the condition of r, which is $front^1 \geq 15 \wedge front^2 \geq 15$, where $front^1$ and $front^2$ correspond to the sensing variable $front$ in rule r_0 from M^1 and M^2, respectively. Both $front^1$ and $front^2$ are affected by uncertainty, without losing generality, we use $front^1$ for illustration. The error ranges for $front^1$ is $[-6, 6]$. We use a new variable $front'^1$ to replace $front^1$ in the path condition. The constraint between $front^1$ and $front'^1$ is "$front^1-6 \leq front'^1 \leq front^1+6$". Variable $front^2$'s constraint with new variable $front'^2$ is similar. Then we combine all these new constraints into the path condition, and get a new condition of σ, i.e., "$front'^1 \geq 15 \wedge front'^2 \geq 15 \wedge front^1 - 6 \leq front'^1 \leq front^1 + 6 \wedge front^2 - 6 \leq front'^2 \leq front^2 + 6$ ".

4.2 Modeling the Environment

A typical CPS application's execution closely interacts with the environment. To make an appropriate action, the application needs to gather information by sensing its environment. For the purpose of verification, it is necessary to model the environment, since different environments would result in different application behaviors. The environment can be very complex, and may contain tremendous data. However, one only needs to model the attributes related to the application. We propose to model these attributes with a tuple of variables as $E = (e_1, e_2, \ldots, e_n)$. The selection of the attributes is application specific, and needs to be provided by the user or the verifier. For example, in the robot-cars application, the environment is the area in which the cars are moving. The area has many attributes, such as its temperature, humidity and latitude. However, many attributes are irrelevant, since the cars only care about the positions of the obstacles and checkpoints, which we propose to model by their coordinates. Without losing generality, we assume the obstacles are all cubes, whose height is also irrelevant and thus can be viewed as rectangles. We use the coordinates of the left-up and right-down corners of the rectangles to specify the obstacles' positions. The checkpoints are assumed to be points, and are specified by one coordinate each. Besides, we distinguish the coordinates of the rectangles and the checkpoints by using the first two variables in the tuple to specify the number of the obstacles and the checkpoints, respectively. Take a look at Figure 4, there are two obstacles and two checkpoints, and their coordinates are labelled in parentheses. So the environment E is (2, 2, 20, 90, 60, 80, 60, 60, 80, 30, 10, 20, 90, 90). Here,

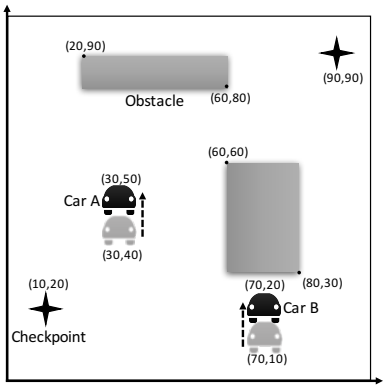


Fig. 4. An environment setting scenario for robot-cars application.

we used one tuple to model the environment, which is sufficient for specifying a static environment as in our example. To support the modeling of dynamic environments, one could use a sequence of tuples and associate each tuple to one or more actions. Each tuple models the status of the dynamic environment at that moment when the tuple's associated actions are taken.

Although we have successfully modeled the environment, it is not enough to perform the verification, since this "complete" model is different than the one sensed by the application. The application can only sense the environment partially, which requires the mapping between the complete environment model E and the partial environment model P. Furthermore, the application's behavior is dynamic, which results the partial model constantly changing, even the complete environment is static. We propose a function F to characterize the mapping between E and P, and since F must reflect the changing nature of the partial model, it also takes the application's current state S as an input. To put in simpler words, the function F: $P = F(E, S)$ produces the application's current understanding to the environment, and therefore its semantics is defined by the specific application and environment. Take the robot-cars application for example again. Suppose Car A first is at location $(30, 40)$ (marked in grey), and it makes a "move" action for 10cm and updates its current state. In this new state the car's position is $(30, 50)$. The function F takes the car's current position and the obstacles' positions specified in the environment model E, produces the car's front, left, right and back distance to obstacles by applying Euclidean geometry, which are 30cm, ∞, 30cm, ∞, respectively. Similarly, the front, left, right and back distance for Car B can be calculated, which are 10cm, ∞, ∞, ∞, respectively. Also, we assume that a checkpoint is visited if the car is within the radius of 5cm of the checkpoint. In this example, both checkpoints have not been visited.

Now we can augment the path condition of a selected path with such environmental settings, as mentioned in the overview. For each rule in the path, we apply function F on the rule's source state and the environment model E, to get the values of the unprimed sensing variables. After it is done, we get a new formula of constraints for the path. For example, for the same path $\sigma = (A^1A^2)\ r\ (A^1A^2)$ used in the last subsection of the robot-cars example, suppose the application's current state is the first A^1A^2 in σ and in this state the two cars' positions are shown in Figure 4 in black. Then for the condition "$front'^1 \geq 15 \wedge front'^2 \geq 15 \wedge front^1 - 6 \leq front'^1 \leq front^1 + 6 \wedge front^2 - 6 \leq front'^2 \leq front^2 + 6$ ", we have $front^1 = 30$ and $front^2 = 10$, since $front^1$ and $front^2$ are variables reflecting the actual environment. After substituting $front^1$ and $front^2$ with concrete values, we get a new formula which is only about the primed variables. The augmentation

of the failure condition is the same as the path condition, so the repetitious details need not be given here.

4.3 Prioritizing Counterexamples

To check whether an execution can lead to application failures, we concatenate a failure condition to the path condition associated with this execution to get a new constraint, which is then passed to Z3 to check whether there is a satisfying solution. If yes, the execution can fall into failure. In the solution, for each variable, Z3 will provide it with a value. A path σ contains a sequence of actions that the application takes under a specific environment, which would lead to the application's failure if a solution exists. The path σ, the environment E and the solution together indicate an application failure, and that is why we name it a counterexample. A counterexample is a tuple $t = (\sigma, E, L)$, where E is the given environment that the application runs in and L is a mapping $L : V \rightarrow A$, where V is a set $\{v_0, v_1, \ldots, v_n\}$ containing all variables in the condition of σ augmented with uncertainty and environmental settings, and A is a set of values $\{a_0, a_1, \ldots, a_n\}$ where $L(v_i) = a_i$ $(0 \leq i \leq n)$ in the solution. A counterexample represents a failed application execution in a specific environment. For example, for the robot-cars application, we already know the environment (cf. Figure 4), the failure condition (cf. Section 4) and the path condition of $\sigma = (A^1A^2)\ r\ (A^1A^2)$, (cf. Subsection 4.2). Then by solving the concatenation of the path condition and the failure condition, we can obtain a solution of "$front'^1 = 32, front'^2 = 15$". This solution corresponds to the scenario as illustrated in Figure 4. The current locations of Car A and Car B are $(30, 40)$ and $(70, 20)$, respectively, and the distances between the cars and their front obstacles are 30 and 10, respectively. If the sensed distances were precise, Car A could move forward but Car B must not do so. However, due to uncertainty, Car A's sensed distance is 32 ($front'^1$) and Car B's sensed distance is 15 ($front'^2$). Based on the sensed distances, rule r in σ is satisfied, and both Car A and Car B made a move action for 10, resulting in that Car B bumps into the front obstacle. This counterexample would not have been reported, if we had not considered uncertainty caused by unreliable sensing.

CPS applications are more likely to fall into failures in an environment with uncertainty [3, 56]. As a result, many counterexamples could be reported. However, for a certain counterexample $t = (\sigma, E, L)$, given the environment E, the application may not execute exactly following the actions specified in σ to incur a failure. This is because each time the application runs, due to uncertainty, the application's sensed environment can be different from its actual environment. The likelihood for application making the same actions as the ones in σ that lead to failure in the environment E is called the probability of counterexample t. We propose to prioritize reported counterexamples to save the effort for fault inspection and fixing. We believe that the more likely a counterexample is to occur, the more attention it needs. Therefore, we rank counterexamples according to their probabilities from high to low.

For a counterexample $t = (\sigma, E, L)$ to occur in the environment E, it requires that the application should make the same sequence of actions as specified in the rules of σ. This means that for each rule r in σ, $r.condition$ should be satisfied so that $r.actions$ can be taken. Thus we first estimate the probability of satisfaction of σ's path condition. We now present how to calculate a counterexample's probability. Let PC be the σ's path condition augmented with uncertainty and environmental settings. In general, the probability $Prob$ that we aim to obtain can be calculated as:

$$Prob = \int_D \mathbb{1}_{PC}(\mathbf{X}) \cdot p(\mathbf{X}) \tag{1}$$

where D is the input domain defined as the Cartesian product of domains of variables in the path condition PC, $p(\mathbf{X})$ is the probability density function of an input $\mathbf{X}$ that specifies the distribution

of $\mathbf{X}$'s value in its error range, and $\mathbb{1}_{PC}(\mathbf{X})$ is the indicator function on PC that returns 1 when $\mathbf{X}$ satisfies PC, and 0 otherwise.

Among variables in input $\mathbf{X}$, some are non-sensing variables and some are sensing variables. The counterexample's probability is irrelevant to the values of non-sensing variables, since they are not affected by uncertainty. Thus, we replace the non-sensing variables in Equation 1 with their values in the counterexample, and only focus on the sensing variables. Still, it can be expensive to directly calculate Equation 1 as a whole. We devised a method to divide Equation 1 into smaller parts and calculate them separately and then merge the results. Prior to this, remind that as discussed in Section 3, the path condition PC of σ is a conjunction of the rules' conditions. Based on this observation that conjunction plays a major role in composing the path condition, we present the following theorem, which divides two conjunctive conditions.

Theorem 4.1. *Let F be a probability function in the form of $\int_D \mathbb{1}_C(\boldsymbol{X}) \cdot p(\boldsymbol{X})$, where D, $\mathbb{1}_C(\boldsymbol{X})$ and $p(\boldsymbol{X})$ are interpreted the same as in Equation 1. Suppose $C = C_1 \wedge C_2$, and $\mathbf{X_1}$ and $\mathbf{X_2}$ contain exactly all the variables appeared in C_1 and C_2, respectively. If there is no common variable x in $\mathbf{X_1}$ and $\mathbf{X_2}$, then*

$$F = \int_{D_1} \mathbb{1}_{C_1}(\mathbf{X_1}) \cdot p(\mathbf{X_1}) \times \int_{D_2} \mathbb{1}_{C_2}(\mathbf{X_2}) \cdot p(\mathbf{X_2})$$

where D_1 and D_2 are the Cartesian product of domains of variables in $\mathbf{X_1}$ and $\mathbf{X_2}$, respectively.

The proof of Theorem 4.1 is straightforward and omitted here. It follows that the calculation of a probability *Prob* can be reduced to the calculation of two smaller sub-functions. Note that clause C_1 and C_2 can be further divided into sub-clauses if they also consist of conjunctive clauses. Therefore, we can repeatedly divide the clause and apply Theorem 4.1 to rewrite the probability function, until no clause is dividable.

Now to calculate probability *Prob*, we just need to calculate every sub-function $\int_D \mathbb{1}_C(\mathbf{X}) \cdot p(\mathbf{X})$ and merge the results. With a reported counterexample $t = (\sigma, E, L)$, since we have replaced the non-sensing variables with concrete values in L and the unprimed sensing variables with values in E, the variables in $\mathbf{X}$ are all sensing variables including uncertainty, i.e. the primed sensing variables. As mentioned earlier, the value of each primed sensing variable v' is the result of the value of its corresponding unprimed variable v plus an error within the error range $[a, b]$. So function $\int_D \mathbb{1}_C(\mathbf{X}) \cdot p(\mathbf{X})$ ($\mathbf{X} = [v'_1, v'_2, \ldots, v'_j]$) can be rewritten into a more concrete form of Equation 2.

$$\int_{v_1+a_1}^{v_1+b_1} \cdots \int_{v_j+a_j}^{v_j+b_j} \mathbb{1}_C(v'_1, \cdots, v'_j) \cdot p(v'_1, \cdots, v'_j) \mathrm{d}v'_1 \cdots \mathrm{d}v'_j \tag{2}$$

For example, we assume that Rule r_0 is the rule in a counterexample's path. The rule's condition $front \geq 15$ involves one variable $front$ affected by uncertainty, and we replace $front$ with $front'$. Suppose the error range of $front$ is $[-6, 6]$, and the value of $front$ in the counterexample is 13. It means that $front'$'s value can range from 7 to 18 due to uncertainty. The distribution of $front'$'s value complies with the Gaussian distribution $N(13, 2^2)$ with a variance of 2^2, and a mean of 13 (the value of $front$). Variable $front'$ could take any value between $(-\infty, +\infty)$ if it followed Gaussian distributions, so we need to set up an error range to restrict the value of $front'$ for the purpose of verification, since $front'_0$ without an error range can be an arbitrary value (e.g., a very large number with an extremely low probability of occurrence) and makes the verification useless by reporting counterexamples at any conditions. In this example, we set up the error range as $[-6, 6]$, so with a mean of 13 the value range of $front'$ is $[7, 18]$, which covers 98.76% of all the possible values of $front'$. Then the probability of $front' \geq 15$ being **true** is calculated according to Equation 1 as $\int_7^{18} \mathbb{1}_{PC}(front') \cdot p(front') \mathrm{d}front'$. Since $\mathbb{1}_{PC}(front') = 0$ when $front' \in [7, 15)$, and

$\mathbb{1}_{PC}(front') = 1$ when $front' \in [15, 18]$, the above equation is equivalent to $\int_{20}^{23} p(front')\mathrm{d}front'$, which results in

$$\int_{15}^{18} \frac{1}{2\sqrt{2\pi}} e^{-\frac{(front'-13)^2}{8}} \mathrm{d}front' = 0.1525.$$

5 CALIBRATING UNCERTAINTY MODELS

The verification approach introduced in Section 4 can verify CPS applications suffering interaction uncertainty and report counterexamples. However, it could be difficult to specify the uncertainty arisen in the interaction between the system and environment precisely and completely because of the uncontrollable noise in the measuring process and insufficient data sample size. Verification results can deviate from reality when the uncertainty specification used in verification is imprecise: less important counterexamples can be falsely highlighted, or even worse, false counterexamples may be reported. This manifests a form of expression that counterexamples' calculated probabilities can be widely divergent from their actual probabilities of occurrence in reality, since uncertainty models are crucial inputs in calculating counterexamples' probabilities. For example, suppose that a counterexample's calculated probability is 0.1, but we did not observe any failure for 100 running times in this counterexample's environment, which is obviously unreasonable and indicates that the input uncertainty models are imprecise. We have conducted a series of experiments to explore the relationship between imprecise uncertainty models and counterexamples' probabilities. The results show that imprecise uncertainty models cause inconsistency between reported counterexamples' calculated and actual probabilities, which is shown in Section 6. Based on this key observation, we propose to identify the imprecise uncertainty models by comparing the differences between selected counterexamples' calculated and actual probabilities.

In this section, we present a search-based approach to calibrating uncertainty models. The intuition behind our approach is that the calibration problem can be formulated as a search problem whose goal is to minimize the difference between counterexamples' calculated and actual probabilities. Another potential solution to obtain more precise uncertainty models is to use statistical methods. For example, one can try to collect a large number of sensors' historical data and generate new more precise uncertainty models by statistically analyzing the data. Compared with this method which demands huge time and manpower to collect "enough" data, our approach does not need large data set to produce new uncertainty models from scratch; it tweaks the existing uncertainty models whenever and whatever amount of data are available, similar to the notion of calibration in mathematics and mechanics. Meanwhile, approaches from the machine learning area can also help on some problems of the kind. For example, linear regression [38] is a simple but widely-used approach for modeling the relationship between a scalar dependent variable and one or more independent variables. Corresponding to our problem, the dependent variable could be the probability of a counterexample, and the independent variables could be variables characterizing the counterexample (e.g., its path condition). The values characterizing the uncertainty model are unknown parameters of the function. However, linear regression is not suitable for our problem, since the relationship between dependent and independent variables is not a linear one, which is specified by a non-linear probability function. Although more complex techniques, e.g., support vector machine [48] and neural networks [42], can be used to model non-linear relationships, a non-negligible challenge still exists. To apply these techniques, a model has to be trained. The acquisition of such a model usually requires tens of thousands of training data, whereas in our problem, the number of the counterexamples is far from meeting this requirement.

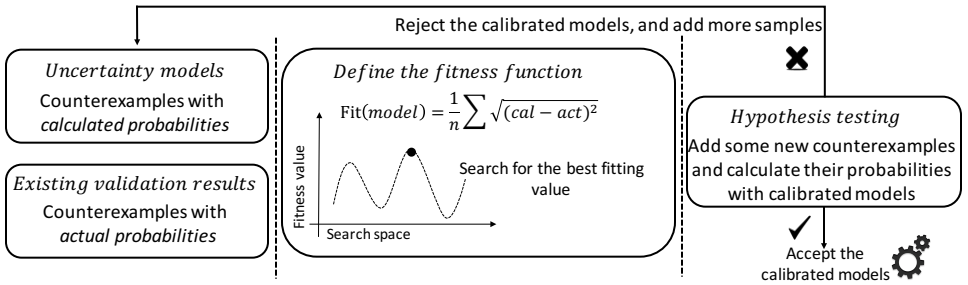


Fig. 5. Overview of calibration process.

We first describe our calibration approach in a nutshell. The calibration problem is represented as an SBSE problem, whose fitness function is to minimize the differences between the counterexamples' calculated probabilities and actual probabilities. The input of the problem is existing uncertainty models and reported counterexamples, and the output is calibrated uncertainty models. The overall of the calibration approach is depicted in Figure 5. It consists of three phases: identification, searching and hypothesis testing. In the identification phase, imprecise uncertainty models are identified if the difference between counterexamples' calculated and actual probabilities is significant (larger than a given threshold). In the searching phase, search-based algorithms is employed to find solutions (calibrated uncertainty models) that minimize the fitness function. Search-based algorithms can provide automated evaluation to reason about the quality of the calibrated uncertainty models based on the existing data set. However, there still exists a chance that these data are not representative, which results in the calibrated uncertainty models over-fitting to specific data and deviate from the precise ones. To tackle this challenge, we employ hypothesis testing to help us decide whether the calibrated models are precise and representative both. If not, we introduce more experiment samples for the calibration and repeat the process until the hypothesis testing validates the calibrated uncertainty models.

5.1 Solving the Calibration Problem

We propose to transform the calibration problem into an SBSE problem. Given a set of counterexamples with actual and calculated probabilities, the search problem starts with the provided uncertainty models and searches for the ones that minimize the fitness function which is about the differences between the counterexamples' actual and calculated probabilities. To be more specific, uncertainty models are characterized by error ranges and distributions which further can be dissected into parameters, e.g., mean and variance for the Gaussian distribution. Therefore, the search problem is to find the values of the uncertainty models' parameters to minimize the fitness function. In the followings, we formulate the fitness function and present the search algorithm.

5.1.1 Definition of the Fitness function. Fitness function describes the objective of the searching problem, and is often built from related metrics or properties. As mentioned earlier, one needs to identify the inconsistency between the selected counterexamples' calculated and actual probabilities. We calculate such differences between two probabilities and use their absolute values' sum divided by the number of counterexamples to specify the degree of inconsistency. Since our goal is to find proper values of error ranges and distributions such that the inconsistency between the selected counterexamples' calculated and actual probabilities is minimized, we directly used it as the fitness functions. To obtain the actual probability of a counterexample, we let the application run in the corresponding environment for a fixed number of times, and count the number of failures

encountered by the application. Then the ratio of the number of failures to the total number of runs serves as the actual probability. Since there could be many counterexamples, which makes it very time-consuming to obtain all the counterexamples' actual probabilities, we need to select some counterexamples to serve as samples. There could be more than one strategy to select counterexamples, but it is recommended to select the counterexamples with high probabilities first to speed up the iterative calibration process.

Let $U = (u_1, u_2, \ldots, u_m)$ be a vector of the variables constituting the uncertainty model, and $\Theta = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ be a set of selected counterexamples. Each $u_i \in U$ $(1 \leq i \leq m)$ could be the lower bound or upper bound of the error range, or the value that characterizes the distribution. For example, a Gaussian distribution is characterized by mean and variance and a uniform distribution is characterized by a range's lower bound and upper bound. For each $\sigma_i \in \Theta$ $(1 \leq i \leq n)$, it has a calculated probability cal_i which is estimated based on the uncertainty model U and an actual probability act_i which is obtained from field study. The fitness function is to measure the differences between probabilities cal_i and act_i. As there are multiple counterexamples, we use the average value of differences to define the fitness function:

$$fitness = \frac{1}{n} \sum_{\sigma_i \in \Theta} \sqrt{(cal_i - act_i)^2},$$

where n is the number of selected counterexamples.

5.1.2 Searching algorithms. There are many excellent search algorithms that appear frequently in the software engineering literature, e.g., Hill Climbing, Simulated Annealing and Genetic Algorithms. These algorithms have different characteristics and can be used in different scenarios. For instance, Hill Climbing and Simulated Annealing are considered to be local searchers, and they operate with reference to one candidate solution at any one time, choosing moves based on the neighborhood of that candidate solution [20, 40]. On the other hand, Genetic Algorithm is said to be a global searcher, sampling many points in the search space at once, offering more robustness to local optima [19]. Therefore, we employ Genetic Algorithms to search for the optimum in our problem.

A Genetic Algorithm is a meta-heuristic search technique that simulates the process of natural selection for solving optimization problems, which was introduced by Holland [23]. In a Genetic Algorithm, each candidate solution is called an individual. There is a fitness function to evaluate how close an individual is to the optimum. The process of Genetic Algorithm typically starts from a set of individuals, randomly selected or pre-defined, to form the first generation. When to create the next generation, all the individuals in the current generation are put into a selection pool, in which each individual has a probability, to be a parent for generating individuals in the next generation. To create an individual in the next generation, a pair of parent solutions are selected from the selection pool. Then, two types of operations (i.e., crossover and mutation) are used to create a child of the two parents. The generated children, optionally plus the individuals from the previous generation, form the next generation. The selection-creation loop is repeated until reaching a termination condition, such as finding a good enough solution, or reaching the maximum number of generations. There are several key components in designing a genetic algorithm, such as the selection method (i.e., how to select individuals for reproduction), crossover operator (i.e., how to produce a child from two parents), mutation operator (i.e., how to mutate a child), and initial population (i.e., how the first generation is populated).

We perform the calibration problem solving with Genetic Algorithm, and the process is shown in Algorithm 2. The input consists of the uncertainty model U, the set of selected counterexamples Θ, an integer n specifying the maximum iteration times and a small real value ϵ serving as the

termination condition of the main iteration in the algorithm. The algorithm then generates an initial population of a fixed size, which could be set to 50 when there are five or fewer variables in the fitness function, or 200 or more depending on the number of variables, as suggested by MATLAB instructions [34]. The population size can also be adjusted with experience. Increasing the population size enables the genetic algorithm to search more points and thereby obtain a better result. However, the larger the population size, the longer the genetic algorithm takes to compute each generation. A common approach to generating an individual in the initial population is to randomly pick a value for each input variable (i.e., each element in U) in the fitness function from the variable's value range. However, this approach ignores the already in-hand uncertainty model which is obtained from statistical analysis, though being imprecise to some extent, but still has more credibility than randomly chosen values. In other words, the existing uncertainty model is imprecise but not wrong, so the precise model in search should be close to the existing one. Therefore, we propose to generate the initial population based on the existing uncertainty model. First, we encode the values of all the variables in the uncertainty model U into a binary form. Then for each char, we randomly change its value (from 0 to 1 or from 1 to 0) at a mutation probability p_m whose value will be discussed later. The core part of the algorithm is the iteration process (Line 5-13), in which the population evolves and individuals are chosen based on their evaluation values. The searching terminates when it reaches the maximum iteration times n or the population's best evaluation value no longer changes significantly (i.e., the difference between best evaluation values of two consecutive iterations is smaller than ϵ). Then the most fitted individual is decoded to obtain the input variables' values, which forms the calibrated uncertainty model.

ALGORITHM 2: Genetic Algorithm based calibration

Input: the uncertainty model U, the set of selected counterexamples Θ, the iteration times n, and the difference ϵ between two consecutive evaluation values.
Output: the calibrated model s.

```
1:  fit := the fitness function;
2:  pop := generateInitialPopulation(U); //pop is a vector representing the population
3:  evalValue := 0;
4:  k := 0;
5:  repeat
6:      evalValue' := evalValue;
7:      fitnessValue := evalFitness(fit,Θ, pop); //fitnessValue is a vector
8:      evalValue := min(fitnessValue);
9:      pop := select(pop,fitnessValue);
10:     pop := crossOver(pop);
11:     pop := mutate(pop);
12:     k++;
13: until k < n or evalValue − evalValue' < ϵ
14: individual := selectMostFitted(pop);
15: s := decode(individual);
16: return s;
```

During the iteration process, there are several key operations: selection (Line 9), crossover (Line 10) and mutation (Line 11). We introduce them briefly in the following:

- Selection. It chooses parent individuals for the next children population based on the scaled values from the evaluation. For each individual i, its evaluation value $eval_i$ is the value of the fitness function fit given individual i as the uncertainty model. We adopt a standard

selection method, i.e., roulette wheel selection, where the probability of selecting individual i is $eval_i / \sum_{j=1}^{m} eval_j$, where m is the total number of individuals. The selected individuals are in the selection pool.

- Crossover. It crosses two parent individuals to form two new children individuals for the next population. To determine the parents for crossover, given a crossover probability p_c, we randomly pick $p_c * m/2$ parents from the selection pool. In a specific crossover operation, single point strategy is applied to cross the parents. The parents are then replaced by their children in the selection pool.
- Mutation. It makes small random changes to the individuals of the population, which provides genetic diversity and enables the Genetic Algorithm to search a broader space. We choose the individual in the selection pool one by one, and randomly change its char (e.g., changing a binary form individual's char from 0 to 1) at a mutation probability p_m.

The setting of Genetic Algorithm's parameters (e.g., crossover probability p_c and mutation probability p_m) is problem-dependent. Generally speaking, increasing mutation probability can increase the genetic diversity, but will cause the searching too scattered over the solution space, while increasing crossover probability can lead the algorithm to search in a relatively concentrated area over the solution space. We could also adjust the parameters based on the experiment feedbacks. Meanwhile, existing literature has summarized many valuable experiences for our reference in dealing with different problems [53]. As recommended, it is safe to set the crossover probability from $[0.4, 0.9]$ and mutation probability from $[0.0001, 0.1]$. The solution includes the values of the variables which represent the calibrated uncertainty model.

5.2 Hypothesis testing

After we calibrate the uncertainty models, it is still a problem to determine whether the calibrated models are precise enough or not, since though the fitness function can evaluate the quality of the calibrated model on the existing data sample, the model may over-fit to these data. To tackle this challenge, we introduce a new sample of data and employ hypothesis testing to help us decide whether there is enough evidence in this new sample of data to infer that the uncertainty model is precise for the entire population. A hypothesis testing examines two opposing hypotheses about a population: the null hypothesis (H_0) and the alternative hypothesis (H_1). The null hypothesis is the statement being tested. Based on the test statistic obtained from the sample data, the testing determines whether to accept or reject the null hypothesis. For our problem, the question we want to answer with hypothesis testing is whether the calibrated uncertainty models are precise enough. As mentioned earlier, it is reasonable to believe that there should be no significant differences between counterexamples' calculated probabilities and actual probabilities when the uncertainty models are precise. With this observation, we can propose our null hypothesis H_0 and alternative hypothesis H_1. For a set of counterexamples which have not been used in calibration $\sigma_1, \sigma_2, \ldots, \sigma_n$, we calculate each counterexample σ_i $(1 \leq i \leq n)$'s probability cal_i with the calibrated uncertainty model and obtain its actual probability act_i in field study, which results in n pair independent observed data: $(cal_1, act_1), (cal_2, act_2), \ldots, (cal_n, act_n)$. Let d_i $(1 \leq i \leq n)$ be the difference between σ_i's calculated and actual probabilities, i.e., $d_i = cal_i - act_i$. Since each d_i is the expression of the same cause that the calibrated uncertainty model deviates from the precise one, with statistical theory it is safe to assume that they are subject to the same distribution, which is often a Gaussian distribution. Assume $d_i \sim N(\mu_d, \sigma_d^2),\ i = 1, 2, \ldots, n$, which means $d_1, d_2, \ldots, d_n$ constitute one sample of the normal population $N(\mu_d, \sigma_d^2)$ where μ_d, σ_d^2 are unknown. Based on this sample, we need to test our hypotheses $H_0 : \mu_d = 0$ and $H_1 : \mu_d \neq 0$. H_0 represents that there is no significant difference between counterexamples' calculated probabilities and actual probabilities, and on the

contrary H_1 represents that there are significant differences between them. In the following, we need to test whether we can accept H_0. If so, we can faithfully conclude that the calibrated models are precise enough and terminate the calibrating process. Otherwise, we introduce more experiment samples for calibration and repeat the calibrating process until the hypothesis testing validates the calibrated uncertainty models.

We adopt the frequently-used p-value method in hypothesis testing. The p-value p is defined as the probability, under the assumption of the null hypothesis, of obtaining a result equal to or "more extreme" than what was actually observed. In this method, we first choose a threshold value for p for the null hypothesis H_0, called the significance level of the test, traditionally 5% or 1% and denoted as α. If the p-value is less than or equal to the chosen significance level (α), the test suggests that the observed data is inconsistent with the null hypothesis, so the null hypothesis must be rejected. Otherwise, the null hypothesis is accepted. For example, using the standard $\alpha = 0.05$ cutoff, the null hypothesis is rejected when $p < 0.05$ and not rejected when $p > 0.05$. Hence, the computation of the p-value is a key step in our hypothesis testing, and to do so we first need to identify a test statistic. There are many test statistic (e.g., paired tests, Z-tests, t-tests and Chi-squared tests), and the selection of the test statistic is problem-depended. For example, Z-tests are appropriate for comparing means under stringent conditions regarding normality and a known standard deviation, and t-tests are appropriate for comparing means under relaxed conditions (less is assumed). Using the test statistic's observed value and its known distribution, we can calculate the p-value. For example, to conduct the hypothesis test for the population mean μ in a Gaussian distribution $N(\mu, \sigma^2)$ (σ is unknown), the hypotheses are $H_0 : \mu = \mu_0,\ H_1 : \mu \neq \mu_0$. We use the test statistic $t = \frac{\overline{X}-\mu_0}{S/\sqrt{n}}$ in which $\overline{X}$ is the mean of samples, S is the standard deviation of samples, and n is the number of samples. Suppose the test statistic value t calculated from sample data is t_0, then when t_0 is greater than 0 the p-value equals to $P_{\mu_0}\{|t| \leq t_0\}$, and when t_0 is less than 0 the p-value equals to $P_{\mu_0}\{|t| \leq -t_0\}$. Many existing statistical tools support the automated calculation of p-value.

We can apply the same method to calculate the p-value in our testing. Denote the mean and variance of sample $d_1, d_2, \ldots, d_n$ as $\overline{d}$ and s_d^2. According to the characteristics of the sample, we also use the test statistic $t = \frac{\overline{X}-\mu_0}{S/\sqrt{n}}$, then we substitute the corresponding values and get $t = \frac{\overline{d}}{s_d/\sqrt{n}}$. After that we can calculate the p-value based on the method discussed above and compare it to the significance level α, which is set to 5% in our problem. If the p-value is greater than α, the test suggests that there is no significant difference between counterexamples' calculated probabilities and actual probabilities, so the null hypothesis must be accepted. In this case, we accept that the calibrated uncertainty models and consider them precise enough. If the p-value is less than or equal to α, the test suggests that there are significant differences between counterexamples' calculated probabilities and actual probabilities, so the null hypothesis must be rejected and the uncertainty models need more calibration. This requires new counterexamples and the subsequent processing of their calculated and actual probabilities, which could be time-consuming. Fortunately, in the hypothesis testing we already have a new sample of counterexamples which obviously contain new information as they reject the existing model. Therefore, we directly introduce these counterexamples and recalibrate the uncertainty models.

6 EVALUATION

Our approach mainly includes two parts: verification and calibration, and their relationship is complementary. Verification provides the input for calibration, and the purpose of calibration is to make the verification obtain more accurate results. In this section, we evaluate the effectiveness

and scalability of verification and calibration, respectively. We consider the following research questions:

- **RQ1:** *How does our modeling of uncertainty improve the accuracy of verification of CPS applications?*
- **RQ2:** *How does the bound of path length configured during verification affect the scalability of our approach?*
- **RQ3:** *Can our approach give an accurate estimate of occurrence probabilities for its reported counterexamples?*
- **RQ4:** *Can our calibration approach effectively calibrate the uncertainty models?*
- **RQ5:** *How does the imprecision degree of uncertainty models affect the efficiency of our calibration approach?*

6.1 Experimental design

CPS applications intrinsically combine hardware and physical systems with software, and their execution involves many aspects including hardware, software and physical environment. Thus, to conduct our experiments, we need to make careful experimental designs, as summarized in Table 1. The key of the design is to choose proper experimental application subjects. Specifically, both the selected applications and their running environments need to be manageable, as otherwise it is hard for us to deploy the applications for experiments. Guided by this principle, we selected 15 different robot-cars applications for experiments. Each robot-cars application consists of two robot-cars that explore an unknown area as introduced in the motivating example. These applications were modified and evolved from applications in existing work [55, 56], which were independently developed by different researchers and students in our university during the past five years. They adopt various strategies to control the robot-cars to explore an unknown area based on their collected sensory data. The strategies are different in that they use different conditions for triggering an action (e.g., some are cautious in that they control to turn a robot-car around even with a longer distance to the facing obstacle), or they make different moves when facing the same situation (e.g., when having an obstacle in the front, some are conservative and control the robot-car to move backward, while others can be aggressive and control the robot-car to continue to explore the area by turning left or right). Therefore, these different strategy implementations provide the diversity needed to experimentally validate our approach. They correspond to different ISM models. We listed the number of states and rules for the 15 applications in Table 2, to show the complexity of the case studies.

As shown in Table 1, RQ1 is to evaluate the effectiveness of our verification approach. The experiment is designed as performing our approach and the approach ignoring uncertainty on 15 robot-cars applications, and observing whether our approach give more real counterexamples. RQ2 is to evaluate the scalability of our verification approach. The experiment is to change the bound of the path length and observe the time and memory cost, when performing verification on the 15 robot-cars applications. RQ3 is to evaluate whether our approach can produce accurate estimation of the counterexample probabilities or not. We choose 10 counterexamples from each of the 15 robot-cars applications as experimental subjects, and the experiment is to observe whether our approach can give accurate probability estimation for the 150 counterexamples. RQ4 is to assess the effectiveness of the calibration approach. The subject is a randomly selected application. In the experiment, we will use uncertainty models of different imprecision degrees, and evaluate the calibration effectiveness of different approaches. Finally, RQ5 is to evaluate the impact of the imprecision degrees of uncertainty models on the efficiency of our calibration approach. We will randomly choose an application as the experimental subject. In the experiment, we will use

Table 1. Designs for the experiments answering the five research questions

RQ	Subjects	Independent variables	Dependent variables	Experimental tasks
RQ1	15 robot-cars applications	verification approach	counterexample	apply and compare different verification approaches
RQ2	15 robot-cars applications	the bound of path length	time and memory cost	set different path length bounds and compare the verification cost
RQ3	150 counter-examples	uncertainty model	calculated probability	compare calculated probabilities with actual probabilities
RQ4	a random application	imprecise model & calibration approach	calibrated model	apply and compare different calibration approaches
RQ5	a random application	degree of model's imprecision	evaluation times	assess the evaluation times in calibrating imprecise models

Table 2. The basic information of the 15 subject applications

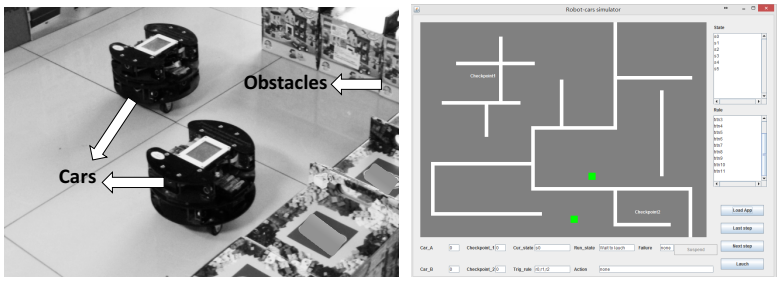
App	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
States	12	5	6	20	6	10	8	15	10	12	22	15	8	7	16
Rules	23	14	12	35	15	21	18	25	29	26	41	30	25	22	23

uncertainty models of different imprecision degrees, and observe how the imprecision degrees will affect the evaluation times in the calibration process. More details will be given when addressing each research question in the following sections.

6.2 Verification

Verification of CPS applications aims at providing evidence that whether the applications would fail or not during operation. This section reports three different experiments we conducted on robot-cars applications to evaluate our proposed verification technique. The first experiment evaluates the approach's failure discovering ability by comparing the verification results of our approach and another one that does not consider uncertainty in the verification. The second experiment evaluates the scalability of our verification approach with different bounds of path length configured. The third experiment assesses whether our approach can estimate counterexamples' occurrence probabilities accurately.

6.2.1 RQ1: Effectiveness of verification. To answer our research question RQ1, we conducted experiments to study whether modeling uncertainty can help improve the verification results. We implemented an approach *naïve*, which ignores uncertainty in the verification, for comparison. Its implementation was the same as our verification approach except that it does not use the method introduced in Section 4.1, which deals with uncertainty. Specifically, our verification approach introduced a new primed variable to each variable affected by uncertainty in the path condition for modeling uncertainty, while for the approach *naïve*, such primed variables for addressing uncertainty are not introduced to the path condition. We applied both our approach (named *actual*) and the *naïve* approach to the 15 robot-cars applications, and recorded their reported counterexamples. We set up a real environment in the lab (Figure 6 (a)) and developed a simulator (Figure 6 (b)) for validating the reported counterexamples. For field tests, we ran robot-cars applications and monitored their behavior. For simulation, we simulated the environmental settings and the



(a) Robot-cars (b) Simulator

Fig. 6. Robot-cars and simulator used in the experiments.

interaction uncertainty with error ranges and distributions. We ran the 15 robot-cars applications in both real deployments (i.e., field study) and simulation. For all the confirmed counterexamples, we compare the two approaches to see whether there is any counterexample reported by *actual* but not by *naïve*, and vice versa. In other words, our goal is to study whether *actual* can report more counterexamples than *naïve*.

We examined the effects of modeling uncertainty by comparing the verification results of approach *naïve* and our approach *actual*. We verified the 15 robot-cars applications with both approaches, with the bound of path length set to 50. The results are shown in Table 3. Row 2 is the number of reported counterexamples of *actual*, and each application reported a different number of counterexamples. As discussed earlier, to control the robot-cars, the applications adopt different strategies, which have an impact on their verification results. Applications such as App 1 and 2 that adopt cautious conditions for triggering actions (e.g., a safer distance to a facing obstacle for triggering a turn-around action is longer) reported less counterexamples than those adopting less cautious conditions such as App 7 and 13. Besides, applications that make conservative moves (e.g., when having a facing obstacle in the front, the action is to move backward instead of turning left or right) also reported less counterexamples, e.g., App 3 and 4. Row 3 is the number of reported counterexamples of *naïve*. Our approach *actual* reported clearly more counterexamples than *naïve*. Furthermore, by careful examination we found that all the counterexamples reported by *naïve* are also reported by *actual*. In the meantime, we confirmed that all counterexamples reported by *actual* can happen in real environments. Most confirmations were acquired by field study (more than 82%). The others (less than 18%) were acquired by simulation, because these counterexamples have a fairly low probability to occur, and therefore are difficult to be witnessed in the field study. The experimental results show that our approach *actual* reported 1.6-9.0x more counterexamples than approach *naïve*, which demonstrates that our approach has a better accuracy (more complete). Based on the above discussed experimental results, we derive our answer to research question RQ1: *modeling uncertainty can considerably improve the accuracy of verification of CPS applications.*

6.2.2 RQ2: Scalability of verification. The bound of path length is a parameter in our approach which can affect the approach's performance and the number of reported counterexamples. Besides, since our approach takes one path for verification at a time, the maximum bound of the path, instead of the whole size of the ISM, affects the scalability of the approach virtually. Therefore, we need to investigate how the bound impacts our approach. In particular, we want to know how our approach scales and how the number of counterexamples grows as the bound increases. So, to answer our research question RQ2, we applied our approach to the 15 robot-cars applications to

Table 3. Comparison of experimental results (reported counterexamples) between *actual* and *naïve*

Application	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
Actual	10	19	21	17	24	29	55	37	49	34	18	29	62	38	25
Naïve	1	4	8	4	6	10	11	9	12	5	4	8	14	8	5
More	9	15	13	13	18	19	44	28	37	29	14	21	48	30	20
Improve	9.0	3.8	1.6	3.3	3.0	1.9	4.0	3.1	3.1	5.8	3.5	2.6	3.4	3.8	4.0

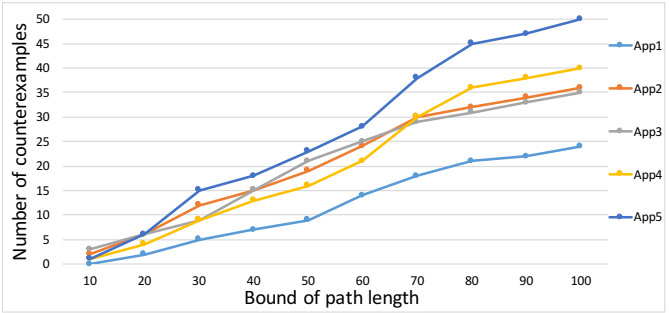


Fig. 7. The growing trend of the number of counterexamples with the increase of the bound of path length.

measure the performance in turns of time and memory costs, and recorded the number of reported counterexamples as the bound increased.

We ran our approach to verify the robot-cars applications with different bounds of path length. The experimental platform is a Dell Desktop PC with an Intel Core(TM) i7 CPU @3.4GHz and 8GB RAM. We recorded the spent time and memory in each run, which are shown in Table 4. As we can observe from the table, when the bound is set to 100, our approach spent about 1 hour and consumed around 700 MB memory on average. Clearly, given enough time, our approach is able to handle larger bounds. However, a bound of 100 is already sufficient for the robot-car applications, if we focus on counterexamples with relatively high probabilities. This is because we observed that when the bound was set to 100, the increase of the number of counterexamples with a probability higher than 10^{-5}, which is a very small likelihood for a counterexample to occur, tends towards stability. Figure 7 illustrates the number of counterexamples with a probability higher than 10^{-5} when the bound was set from 10 to 100 (for App1 to App5). From these discussions, we can derive our answer to research question RQ2: *our approach can scale well to large bounds of path length and it can detect more counterexamples when the bound increases.*

6.2.3 RQ3: Probability estimation accuracy. Through research question RQ3, we want to validate that our approach can give an accurate estimate of occurrence probabilities for its reported counterexamples. But we notice that besides the calculation method, the precision of uncertainty model can also affect the calculation results of counterexamples' probabilities. If the uncertainty model is imprecise, it will lead to inaccurate calculation results. So to answer research question RQ3, we conducted two groups of control experiments, to separate the effects of the calculation method and the uncertainty models. One group was conducted with precise uncertainty models, and the other was with imprecise uncertainty models. Both experiments need the actual probability

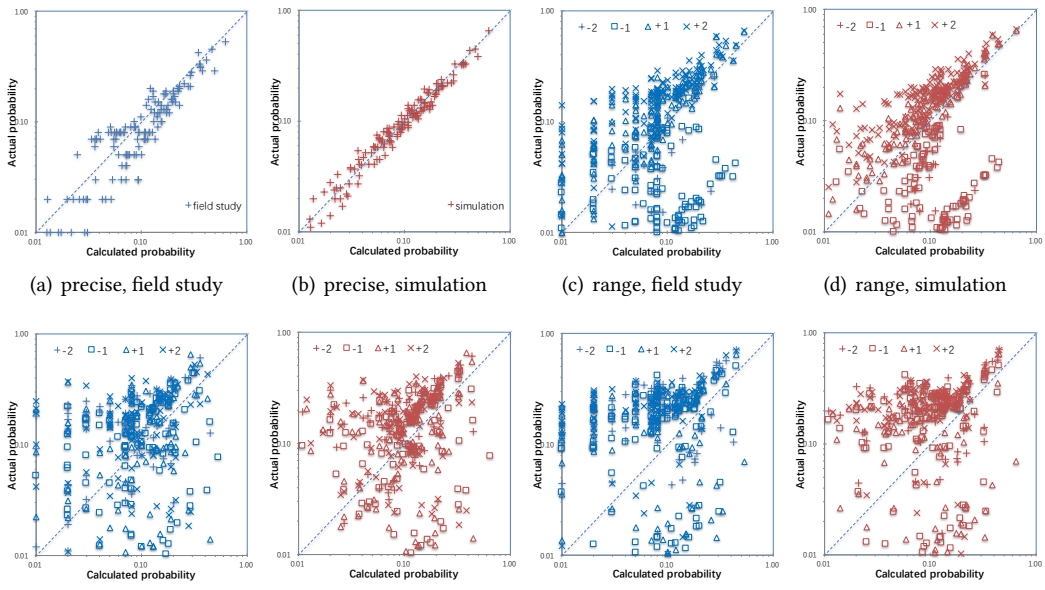
Table 4. Time (second) and memory (MB) costs for verification with different bounds

Application	10	20	30	40	50	60	70	80	90	100
App1	18.0 54	57.3 98	108.9 117	224.5 144	497.6 202	553.1 290	966.4 475	1907.4 504	2697.1 577	3523.7 620
App2	19.7 53	45.5 87	76.2 120	133.8 174	213.5 212	448.1 280	503.2 411	776.4 554	1034.3 633	1920.6 709
App3	12.6 65	33.1 84	63.4 109	99.3 170	156.8 234	278.2 376	530.7 433	902.5 503	1667.0 596	2983.2 690
App4	45.2 156	76.3 267	174.7 413	368.2 540	622.5 689	950.1 804	1730.6 932	2179.4 1108	3157.5 1230	4119.1 1412
App5	10.3 91	36.6 143	128.8 168	344.0 299	595.5 353	821.4 520	1192.8 599	2046.2 652	2899.5 785	3367.9 869
App6	18.2 55	45.6 103	104.3 198	201.3 275	346.6 402	605.4 470	911.3 554	1670.6 643	1929.2 702	2752.7 791
App7	10.5 55	35.7 69	67.9 126	105.5 157	264.6 202	466.1 331	609.4 476	1095.4 547	1719.1 674	2364.7 711
App8	25.1 73	68.3 106	119.5 197	235.6 253	515.2 301	703.4 422	1023.8 530	2001.3 679	2745.8 744	3520.2 803
App9	22.4 67	56.1 142	106.4 256	175.3 373	533.0 485	883.7 529	1443.3 653	2069.1 788	2921.5 890	4367.7 944
App10	15.4 40	48.6 76	75.1 98	109.4 152	238.8 278	489.2 346	787.0 476	1338.6 533	2030.5 604	3222.1 695
App11	39.0 51	89.0 92	165.9 154	310.7 290	578.1 401	738.6 574	1221.4 632	1903.2 782	3660.5 956	5021.6 1108
App12	28.6 132	57.7 243	110.2 321	208.5 470	312.2 611	555.9 708	987.5 874	2012.8 950	3190.0 1038	4080.6 1221
App13	19.3 54	56.1 96	77.2 125	118.0 174	301.4 208	694.8 290	1026.3 421	1877.5 556	2919.6 643	3797.9 702
App14	15.7 67	76.0 119	126.6 216	224.2 326	568.5 420	801.4 592	1010.8 656	1992.2 771	2702.5 858	3309.9 902
App15	26.9 48	66.8 61	150.6 84	288.6 125	475.2 201	688.1 299	994.0 376	1776.5 565	2107.9 663	3662.5 746

* In every cell with two data, the upper one is the time and the lower one is the memory.

of every counterexample, i.e., the likelihood of occurrence of a counterexample in its corresponding environment, to serve as the ground truth to assess our predicted occurrence probability. So we first let an application run in the corresponding environment of every counterexample, and counted the number of failures encountered by this application with respect to its corresponding counterexample. The experiment was conducted for top 10 reported counterexamples of each of the 15 robot-cars applications in both field study and simulation. Specifically, for each selected counterexample, we let the concerned application run in the corresponding environment of this counterexample in field study for 100 times, and also in simulation for 1,000 times.

For the first group of experiment, we need to obtain precise uncertainty models for the robot-cars applications. We studied interaction uncertainty occurring in the robot-cars applications and analyzed many sample data. Meanwhile, we leveraged the uncertainty models that have already been confirmed to be considerably precise by existing work [56], and used them as the precise models. Based on these models, we calculated the counterexamples' probabilities, and compared



(a) precise, field study (b) precise, simulation (c) range, field study (d) range, simulation

(e) mean, field study (f) mean, simulation (g) variance, field study (h) variance, simulation

Fig. 8. Comparisons of counterexamples' calculated probabilities and actual probabilities. The actual probabilities are either from field study shown in blue marks in Subfigures (a), (c), (e) and (g), or from simulation shown in red marks in Subfigures (b), (d), (f) and (h). The calculated probabilities are estimated based on precise model (Subfigure (a) and (b)), model with imprecise error range (Subfigure (c) and (d)), model with imprecise mean (Subfigure (e) and (f)), and model with imprecise variance (Subfigure (g) and (h)).

them with actual probabilities from field study and simulation, which are shown in Figure 8 (a) and (b) by blue and red marks, respectively. The horizontal axis represents calculated probabilities from our approach, and the vertical axis represents actual probabilities observed from field study or simulation. Both axes are in logarithmic coordinates. It is clear that for a counterexample, if its calculated probability from our approach is close to its actual probability, its corresponding point in the figure would be close to the diagonal line. As we can observe from Figure 8 (a) and (b), most of the points are scattered near the diagonal line. From these results, we can derive a conclusion that *given precise uncertainty models, our approach can accurately estimate occurrence probabilities for its reported counterexamples.*

For the second group of experiment, we generated a series of imprecise uncertainty models and observed their impacts on the estimation results. For the robot-cars applications, the uncertainty model is caused by unreliable sensing, which demonstrates a regular pattern in turns of an error range with a Gaussian distribution. We generated the imprecise models by applying various changes to the precise model. Specifically, we identified three controlled variables: the error range, the mean and the variance of the distribution. The change of these variables was made in a controlled way that each time we changed one variable. To evaluate the degree of the imprecision, we tried both increasing and decreasing the variables by ±1 and ±2. Note that when the error range is changed, we need to first repeat the verification process on the counterexample's path since the changed error range will affect verification results. As mentioned earlier, we speculated that with imprecise uncertainty models, the calculated probabilities would deviate from the actual ones. In theory,

by changing either the error range, mean or variance, the probabilities that variables take values causing counterexamples would change as well. As we can see from Figure 8 (c)-(h), when we changed the error range, mean and variance, there were clear differences between the calculated and actual probabilities, both in field study and simulation.

Based on the above discussed experimental results, we can conclude that *imprecise uncertainty models can lead to inaccurate estimates of occurrence probabilities in our approach*, consistently for all the counterexamples in the experiments. This means imprecise models can be easily discovered, as we only need to sample a few counterexamples, preferably the ones with high probabilities, to compare their calculated and actual probabilities. Once imprecise models are identified, our approach offers the calibration method, which produces more precise models to refine the verification results.

6.3 Calibration

Calibration of uncertainty models aims to further improve precision of the models and consequently the accuracy of verification results. We report results on two experiments in this section. The first experiment examines our calibration approach's effectiveness in calibrating imprecise uncertainty models, and the second experiment evaluates the efficiency of our calibration process.

6.3.1 RQ4: Effectiveness of calibration. This section evaluates the effectiveness of our approach in calibrating imprecise uncertainty models of CPS applications. Our subjects were also the robot-cars applications. We used the imprecise uncertainty models introduced in RQ3 to assess our calibration approach. The uncertainty models were generated from a precise uncertainty model with various changes. We conducted a series of experiments to check whether our approach can successfully calibrate those imprecise uncertainty models. Specifically, we calibrated the imprecise uncertainty models based on the robot-cars applications' counterexamples, each of which has a calculated probability and an actual probability. The calculated probability was calculated with the imprecise uncertainty models, and the actual probability was obtained from field study. Those imprecise uncertainty models together with the counterexamples were fed to our calibration approach.

Table 5 reports our experimental results on a randomly picked application (App 5). In this experiment, we employed the counterexamples' actual probabilities obtained from field study for calibration. The precise uncertainty model, which has the error range $[-6, 6]$ and the Gaussian distribution $N(0, 2^2)$, servers as a baseline to evaluate the precision of the calibrated models. Column 2 shows the calibrated uncertainty model. To compare two uncertainty models, we use the Euclidean distance to represent the difference between the two models. The error range's lower bound l, upper bound u, mean n and variance v constitute the four dimensions of the model m, and the Euclidean distance d between two models m_1 and m_2 is $\sqrt{(l_1 - l_2)^2 + (u_1 - u_2)^2 + (n_1 - n_2)^2 + (v_1 - v_2)^2}$. Then we propose to evaluate the closeness of two uncertainty models with the function $c = \frac{1}{1+d}$. Clearly, c has a value from $(0, 1]$, and the smaller the distance between two models is, the larger c gets. If two models are exactly the same, then $c = 1$. From Table 5's Column 3 we can see that the closeness between imprecise models and precise models ranges from 0.077 to 0.500 (on average 0.309), depending on the degrees of changes we made. The calibrated models, on the other hand, present themselves with a considerably improvement of the closeness to the precise model (Column 4). Ranging from 0.592 to 0.772, the average closeness is 0.681, which has been improved for 120.3% and is close to 1 indicating that the calibrated model's precision has been improved approximately to the level of the precise model. We also recorded the iterating times of fitness function evaluation and total time consumed by the Genetic Algorithm search, which are shown in Column 5 and Column 6, respectively. Hypothesis testing is leveraged to help us decide whether the calibrated uncertainty models are precise and representative. If the test is not passed, we introduce 10 more experiment samples each time for calibration and repeat the process, which will increase the time

Table 5. Calibration results using data from field study

Degree of change	Calibrated model	Closeness (imprecise)	Closeness (calibrated)	No. of eval.	Search time	No. of tests
Range-1	$[-5.86, 5.80],\ N(0.08, 1.88^2)$	0.414	0.652	228	5.38s	1
Range-2	$[-5.76, 5.82],\ N(0.05, 2.11^2)$	0.261	0.647	651	19.67s	3
Range+1	$[-6.12, 6.10],\ N(0.08, 1.88^2)$	0.414	0.668	439	13.07s	2
Range+2	$[-6.18, 6.14],\ N(-0.08, 2.13^2)$	0.261	0.629	511	17.42s	2
Mean-1	$[-6.15, 5.78],\ N(-0.10, 1.91^2)$	0.500	0.688	480	21.05s	1
Mean-2	$[-6.25, 5.75],\ N(-0.12, 1.85^2)$	0.333	0.592	705	25.54s	2
Mean+1	$[-5.80, 6.12],\ N(0.07, 2.06^2)$	0.500	0.744	819	28.72s	2
Mean+2	$[-5.78, 6.17],\ N(0.09, 1.99^2)$	0.333	0.772	954	37.32s	3
Variance-1	$[-6.05, 6.10],\ N(0.13, 1.93^2)$	0.250	0.755	178	6.56s	1
Variance-2	$[-5.86, 5.91],\ N(0.15, 1.88^2)$	0.200	0.659	410	18.04s	3
Variance+1	$[-6.05, 6.10],\ N(0.13, 1.93^2)$	0.167	0.755	276	11.25s	1
Variance+2	$[-5.84, 5.88],\ N(-0.15, 2.14^2)$	0.077	0.613	623	24.33s	4

* All the changes are made on the basis of the precise model: error range $[-6, 6]$, Gaussian distribution $N(0, 2^2)$.

Table 6. Calibration results using data from simulation

Degree of change	Calibrated model	Closeness (imprecise)	Closeness (calibrated)	No. of eval.	Search time	No. of tests
Range-1	$[-5.92, 6.04],\ N(0.06, 2.07)$	0.414	0.767	456	13.86s	2
Range-2	$[-5.86, 5.91],\ N(0.10, 1.95)$	0.261	0.783	632	19.31s	3
Range+1	$[-6.05, 6.13],\ N(-0.05, 2.13)$	0.414	0.642	210	5.04s	1
Range+2	$[-6.14, 6.17],\ N(-0.04, 2.18)$	0.261	0.560	801	22.46s	4
Mean-1	$[-5.94, 5.98],\ N(0.95, 1.93^2)$	0.500	0.777	441	19.83s	2
Mean-2	$[-6.14, 6.07],\ N(0.92, 2.10^2)$	0.333	0.692	676	23.45s	3
Mean+1	$[-5.96, 6.06],\ N(1.11, 1.95^2)$	0.500	0.808	402	18.37s	2
Mean+2	$[-5.98, 6.03],\ N(1.20, 2.04^2)$	0.333	0.794	901	34.58s	3
Variance-1	$[-6.10, 5.84],\ N(0.04, 2.90^2)$	0.167	0.617	266	10.88s	1
Variance-2	$[-6.17, 5.71],\ N(-0.01, 2.79^2)$	0.111	0.442	618	24.02s	3
Variance+1	$[-5.78, 6.10],\ N(0.06, 3.08^2)$	0.125	0.646	447	20.53s	2
Variance+2	$[-5.94, 5.88],\ N(-0.05, 3.15^2)$	0.059	0.517	884	28.49s	3

* The first four changes are made on a precise model: error range $[-6, 6]$, Gaussian distribution $N(0, 2^2)$, the middle four are made on a precise model: error range $[-6, 6]$, Gaussian distribution $N(1, 2^2)$, and the last four are changed on a precise model: error range $[-6, 6]$, Gaussian distribution $N(0, 3^2)$.

consumption. However, as shown in Column 7, our approach just needs few times of hypothesis testing. This conclusion has also been consistently conformed in the simulation. Table 6 reports the calibration results by using counterexamples' actual probabilities obtained from simulation. The benefit of using simulation is that we can easily simulate different kinds of precise uncertainty models and check whether our approach can calibrate their imprecise models effectively. As we can see from Table 6, for different precise models with different mean and variance our approach can effectively calibrate their imprecise uncertainty models.

As introduced in Section 5, Genetic Algorithm is leveraged to solve the calibration problem by searching for the values of the uncertainty model parameters that would minimize the fitness function. Besides Genetic Algorithm, there are alternative options to solve this problem, such as

analytical approaches or other searching algorithms. Therefore, to further evaluate the effectiveness of our calibration approach, we compared our Genetic Algorithm based approach with other approaches. Specially, we selected three other representative algorithms to solve the problem: an analytical algorithm based on trust-region approach [37], the hill climbing [43] and the simulated annealing [27] algorithms. The analytical algorithm is based on the common trust-region approach, which is used in optimization to denote the subset of the region of the objective function that is approximated using a simpler model function. Then through approximation model function and trust region, the original problem is transformed to a simpler trust-region subproblem.

The solver "fminunc" in the MATLAB optimization toolbox offers an implementation of this analytical algorithm, which we leveraged in the experiment. Meanwhile, we implemented the hill climbing and simulated annealing to search the values of the uncertainty model's parameters that minimize the fitness function. The adopted hill climbing is a standard one, and the initial search point was set to the values of original imprecise uncertainty model. For the simulated annealing, besides the initial search point which was also set to the values of original imprecise uncertainty model, there are several other parameters, including initial temperature, the temperature decreasing rate, and iteration times. Guided by existing literature [27], the initial temperature should be high enough, and the decreasing rate should be slightly less than one. Thus we tried several values for the parameters to achieve good performance, and chose 100 for the initial temperature, 0.96 for the temperature decreasing rate and 20 for the iteration times. In the comparing experiments, we just substituted our Genetic algorithm with the other three algorithms. The rest of our calibration method, including the hypothesis test, remained the same.

Again, we used App 5 as the experimental subject, and the precise uncertainty model has an error range of $[-6, 6]$ with a Gaussian distribution $N(0, 2^2)$. The actual probabilities used for calibration is from field study. Table 7 shows the results of the four different algorithms. In each data cell, there are three numbers. The first one is the closeness of the calibrated uncertainty model to the precise one. The second one is the total computing time of solving the problem, and the third one is the number of hypothesis tests. In our approach, when the calibrated result is rejected by the hypothesis test, more counterexamples are needed to repeat the calibration process. The counterexamples and their calculated and actual probabilities are prepared beforehand, however, the repeated calibration process would cost more computing time, which is also included in the total computing time. When the calibration result was rejected for 5 times, the calibration process was aborted. We use the symbol "#" to represent that no calibrated model was obtained. From Table 7, we can see that in general the Genetic Algorithm based approach performed better than the three others, i.e., it computed more precise uncertainty models in less time. More specifically, since the fitness function can be complex by involving probability density functions and integrals, the analytic algorithm's power is limited in sense that it needs to perform complex trust-region computation and factorization. Therefore, it costed more computation time. Hill climbing costed less time in one iteration of search, however, it often fell in local optimum, which makes the calibrated uncertainty model repeatedly rejected by hypothesis tests and the total computation time much longer than our approach. As shown by the experimental results, 5 out of 12 calibrated uncertainty models given by the hill climbing were rejected by hypothesis tests for 5 times. Simulated annealing performed slightly better than hill climbing but still suffered the same problem.

Based on the above discussed experimental results, we derive our answer to research question RQ4: *our Genetic Algorithm based calibration approach can effectively calibrate the uncertainty models.*

6.3.2 RQ5: Efficiency of calibration. In the experiments to answer RQ4, the results showed the efficiency of our calibration approach such that it can complete within a dozen seconds. The data

Table 7. Comparisons of different algorithms for solving the calibration problem

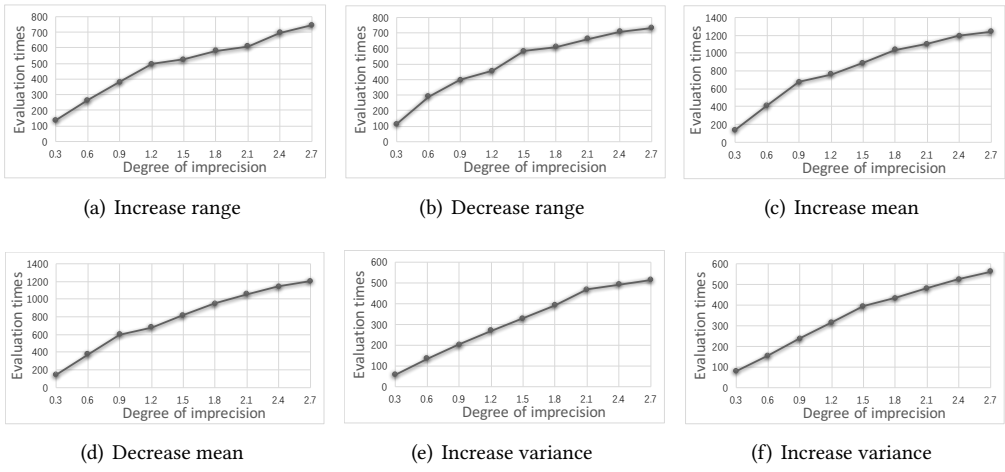
Degree of change	Our approach (closeness/ time/ No. of tests)	Analytic (closeness/ time/ No. of tests)	Hill climbing (closeness/ time/ No. of tests)	Simulated annealing (closeness/ time/ No. of tests)
Range-1	0.652/ 5.38s/ 1	0.624/ 45.21s/ 4	0.602/ 25.18s/ 3	0.697/ 76.58s/ 4
Range-2	0.647/ 19.67s/ 3	0.591/ 60.55s/ 4	#/ 49.06s/ 5	0.579/ 63.49s/ 5
Range+1	0.668/ 13.07s/ 2	0.669/ 64.76s/ 4	0.624/ 50.82s/ 4	0.634/ 64.17s/ 4
Range+2	0.629/ 17.42s/ 2	0.587/ 76.39s/ 4	#/ 71.49s/ 5	0.565/ 43.26s/ 4
Mean-1	0.688/ 21.05s/ 1	0.594/ 83.60s/ 3	0.653/ 85.36s/ 4	0.705/ 58.86s/ 3
Mean-2	0.592/ 25.54s/ 2	0.589/ 70.41s/ 4	#/ 78.43s/ 5	#/ 93.55s/ 5
Mean+1	0.744/ 28.72s/ 2	0.652/ 102.32s/ 5	#/ 104.11s/ 5	0.732/ 84.24s/ 4
Mean+2	0.772/ 37.32s/ 3	#/ 90.52s/ 5	#/ 96.66s/ 5	#/ 120.60s/ 5
Variance-1	0.755/ 6.56s/ 1	0.688/ 40.27s/ 3	0.771/ 36.29s/ 3	0.565/ 53.26s/ 4
Variance-2	0.659/ 18.04s/ 3	0.560/ 63.81s/ 5	0.578/ 125.62s/ 5	0.549/ 84.18s/ 4
Variance+1	0.755/ 11.25s/ 1	0.758/ 76.54s/ 4	0.704/ 45.80s/ 3	0.629/ 62.35s/ 3
Variance+2	0.613/ 24.33s/ 4	#/ 89.46s/ 5	#/ 87.42s/ 5	0.554/ 98.70s/ 5

* All the changes are made on the basis of the precise model: error range $[-6, 6]$, Gaussian distribution $N(0, 2^2)$. Symbol "#" represents that no calibrated model is obtained.

also preliminarily indicate that the imprecision degrees of uncertainty model affect the iterating times of the fitness function evaluation, which constitutes most of the computation time. It seems that more imprecise models tend to require more iterating times than less imprecise models. In this section, we explore how our calibration approach performs with different imprecision degrees of uncertainty models. Specifically, we want to answer: (1) Is there a correlation between the imprecision degrees of uncertainty model and the iterating times of the fitness function evaluation? and (2) Is our approach fast enough to calibrate the imprecise models?

We first randomly selected a robot-cars application (App 11) to conduct the experiments. First, different degrees of changes were made to the application's precise uncertainty model, which has the error range $[-6, 6]$ and the Gaussian distribution $N(0, 2^2)$. We gradually increased and decreased the error range, mean, and variance, respectively, from 0.3 to 2.7 in a step of 0.3. These changes resulted in uncertainty models with different imprecision degrees. Then using our calibration approach, we calibrated these imprecise uncertainty models, and recorded the evaluation times of fitness function. Figure 9 illustrates the experimental results. In each subfigure, the horizontal axis represents the degrees of changes made on different parameters of uncertainty model, and the vertical axis represents the iterating times of evaluation. From this figure we can see that when the imprecision degrees of any parameter increase, the iterating times also increases. However, we can find that the increasing rate of our approach's iterating times of evaluation tends to slow down as the uncertainty models' degrees of imprecision increases. So, even for the maximum change of 2.7 which is a fairly large error for the model under study, the iterating times is just about 500 to 1200 which can be efficiently handled by modern computers.

Based on the above discussed experimental results, we derive our answer to research question RQ5: *the increase of uncertainty model's imprecision degrees can cause an increase of the iterating times of fitness function evaluation, nevertheless, even for a large degree of imprecision, our approach is fast enough to complete the calibration.*



(a) Increase range (b) Decrease range (c) Increase mean

(d) Decrease mean (e) Increase variance (f) Increase variance

Fig. 9. Trends of evaluation times in calibration with uncertainty models of different degrees of imprecision.

6.4 Threats to Validity

There are mainly three kinds of threats: threats to construct validity, threats to internal validity and threats to external validity. We analyze them to the validity of our conclusions below.

Threats to construct validity. The main threat to construct validity for our study is that we may not have run our robot-cars applications adequate times in the field study in order to obtain accurate likelihoods of occurrence of the counterexamples. To reduce this threat, we ran each application in simulation for 1,000 times to get the simulated probabilities as complementary evidence. Another potential threat to construct validity is the threshold of probability used to filter out low-probability counterexamples when evaluating the impact of the bound of path length. Nevertheless, we set the probability threshold to 10^{-5}, which is an extremely small likelihood for a counterexample to occur in real cases. In fact, we believe developers should focus mainly on counterexamples with much higher probabilities in a realistic setting. Finally, a threat to construct validity is that we may not have a sufficient variety of imprecise uncertainty models in evaluating the calibration approach. To reduce this threat, we changed all three parameters in the uncertainty model, by both increasing and decreasing the values in different degrees.

Threats to internal validity. There are mainly two threats to internal validity. One threat to internal validity is the selected bound of path length used in finding counterexamples. It is possible to find more counterexamples with a larger bound. Nevertheless, according to our experiments, the number of counterexamples of high probabilities will tend to be stable as the bound increases. So we selected a properly large value for the bound in experiments. The other threat is that the termination condition for the Genetic Algorithm in calibration may not be strict enough to find an acceptable uncertainty model w.r.t. precision. To reduce this threat, we first chose a fairly strict termination condition; furthermore, we employed hypothesis testing to validate the precision of the uncertainty model.

Threats to external validity. The main threat to external validity is that our conclusions may not generalize to some other CPS applications. Some applications, for example, focus on the temporal aspects of the systems [7, 16]. These applications need the temporal constraints to be satisfied to guarantee correct behavior. As our approach focuses on another important aspect of

interaction between the cyber and physical worlds, it does not explicitly consider the temporal constraints but only supports the temporal synchronization essential for component coordination. Nevertheless, it is possible to consider more temporal properties such as execution-time or deadlines by introducing dense-time clocks to our ISM models. Other applications, with various scales and complexities, may also have their own characteristics. To reduce this threat, we selected 15 different robot-cars CPS applications with various sizes as our experimental subjects. Meanwhile, we made our best efforts in selecting them being independently developed by different developers, and validated by both simulation and field study. The results consistently support our conclusions. Although we have tried to make our approach applicable to other applications, there is still a need for validating our approach with more realistic and larger applications.

7 RELATED WORK

In this section, we discuss some closely related work concerning quality assurance of CPS applications, uncertainty handling, and calibration techniques.

Developing high-quality CPS applications is confronted with stiff challenges, and many research efforts are made to assure their quality. Among them, many studies address the problem of testing CPS applications. Some techniques [3, 47, 57] use model-based testing to generate test cases for CPS, and some work uses runtime monitor to dynamically detect faults in CPS [26]. Simulation is often combined with field testing to test CPS [24, 26, 57], since it can reduce testing cost. Multiple parts of CPS are often embedded systems with a real-time software executing. Therefore, there is some work focusing on testing functional or non-functional correctness (e.g., temperature, power consumption, and timing) of real-time CPS [21, 24, 50]. However, the testing approach is generally infeasible to predict and enumerate all possible conditions that a CPS application can encounter at runtime. Lots of efforts are then spent on developing certifiable verification methods. Formal verification can provide evidence that the set of stated functional and non-functional properties are satisfied during the system's operation [5]. Technically, researchers use model checking [2, 11] or theorem proving [33, 44] to verify CPS. Recent studies have also reported promising results in applying these techniques to verify properties of CPS, e.g., safety [18, 41], robustness [49] and performance [28]. Temporal correctness, as one of the focuses on CPS, has also received much attention. In [16], with the assumption that the constituent parts of the CPS are parameterized with a worst case execution time bound, run-time generation of verified future system configurations is used to guide the evolution of CPS to meet the functional and timing requirements. In [7], an example of a virtualized server node is studied. With the usage of temporal logic of CLTLoc specifying the system behavior, temporal boundaries for the decisions whether a request from a mobile node can be served over the server operation can be provisioned. Different than these researches, our work studies the interactions between the cyber and physical worlds and their effects on the system correctness. When it comes to verifying CPS under physical-world, much existing work lacks of explicitly considering CPS's environmental interaction uncertainty with physical environment.

Uncertainty has always been a bone of contestation in Cyber-physical systems [3] and even the whole software engineering community [13]. Uncertainty poses a big threat to correct and reliable CPS applications. This problem is gaining increasing attention in recent years, but is still relatively unexplored in CPS development and testing. Alippi [4] introduced different sources of uncertainty, which include uncertainty in acquiring data, in interacting with the environment, and in representing information on a finite-precision machine. This echoes our idea of recognizing environment-related interaction as a source of uncertainty for CPS applications. To understand uncertainty in CPS applications better, Zhang et al. [58] derived a conceptual model of uncertainty by reviewing existing work on uncertainty in various domains. The conceptual model is mapped to

the three logical levels of CPS: application, infrastructure, and integration. Besides, they proposed to test CPS applications under unknown risky uncertainty by model-based and search-based testing. Cheng et al. [10] proposed a requirement language RELAX to address uncertainty by enabling software engineers to specify uncertainty in application requirements. The language is in a form of structured natural language with Boolean expressions. The uncertainty is suggested to be identified by a variation of threat modeling, where the threats are the various environmental conditions that pose uncertainty at development time. However, it is difficult to quantitatively model interaction uncertainty caused by unreliable environmental sensing with RELAX for the purpose of system verification.

Our previous work [56] proposed to model interaction uncertainty by error ranges and distributions and leverages an SMT solver to verify the correctness of standalone self-adaptive applications. The current work differs from the existing work at two aspects. First, we studied the problem of verifying CPS applications suffering interaction uncertainty. Since CPS is typically more complex by consisting of multiple components from the physical and cyber worlds, we proposed the new ISM network to model the concurrent and coordinated components in CPS, and devised an approach to encoding the ISM network and the interaction uncertainty to SMT formulae so as to exploit the power of SMT solvers for verification. Second, we noticed the effects of imprecise uncertainty models on the accuracy of verification and validation results, and therefore proposed a search-based approach to systematically calibrating imprecise uncertainty models is proposed, as further discussed below.

Calibration is a general term. In physics it can refer to the act of evaluating and adjusting the precision and accuracy of measurement equipment. In statistics, it can mean a reverse process to regression and procedures in statistical classification to determine class membership probabilities. In our problem it refers to adjusting the parameters of uncertainty models to ensure that the models specify the actual interaction uncertainty as precise as possible (in accordance with the realities). This calibration is necessary because most interaction uncertainties (e.g., environmental sensing) show different characteristics when the CPS applications are configured in a large variety of application domains. One related method is the model calibration, which involves systematic adjustment of model parameters so that outputs more accurately reflect reality [8]. Vanni et al. in article [51] reviewed the common process in model calibration: identifying inputs to calibrate, identifying calibration targets, measuring Goodness-of-Fit (assessing how well outputs match observed data), searching parameters and accepting calibration results. The key parts in the process are the Goodness-of-Fit measures and parameters search strategy. In the literature, the most commonly used measures of Goodness-of-Fit are least squares, chi- squared (or weighted least squares) and the likelihood functions [45]. There are various methods for the solution of different types of searching problems, e.g., Generalized Reduced Gradient Method [29], Simulated Annealing [25] and Genetic Algorithms [23]. Bayesian methods are used in model calibration [17]. They define a prior distribution for the model parameter set, and update the prior via Bayes' theorem to reflect the additional information given in the likelihood function for the data, to give the posterior distribution. To achieve this, an approach known as Markov Chain Monte Carlo can be used to generate a sample from the joint posterior density function of the model parameters [17]. Compared with these pieces of work, our research studies a new problem of calibrating interaction uncertainty models in CPS, and leverages the relationships between the actual and calculated probabilities of counterexamples.

8 CONCLUSION

In this article, we propose a novel approach to modeling and verifying CPS applications with interaction uncertainty. By explicitly taking the interaction uncertainty into account in the verification process, the approach can find many potential counterexamples that would otherwise be overlooked by methods not considering uncertainty. To help developers focus on more likely faults, our approach also prioritizes its reported counterexamples according to their estimate occurrence probabilities. Provided with a precise uncertainty model, our approach can give a good estimate, which has been well validated by both simulated experiments and field study. To deal with the possible imprecise uncertainty model that would jeopardize the accuracy of verification results, we propose a search-based approach to calibrating them. We conducted multiple experiments on real-world applications, and the results consistently confirmed the effectiveness of our verification and calibration approaches.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for helpful comments and suggestions for improving this article. The authors would also like to thank the support from the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

REFERENCES

[1] Sara Abbaspour Asadollah, Rafia Inam, and Hans Hansson. 2015. *A Survey on Testing for Cyber Physical System.* Springer International Publishing, Cham, 194–207. https://doi.org/10.1007/978-3-319-25945-1_12

[2] Ravi Akella and Bruce M. McMillin. 2009. Model-Checking BNDC Properties in Cyber-Physical Systems. In *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09).* IEEE Computer Society, Washington, DC, USA, 660–663. https://doi.org/10.1109/COMPSAC.2009.101

[3] S. Ali and T. Yue. 2015. U-Test: Evolving, Modelling and Testing Realistic Uncertain Behaviours of Cyber-Physical Systems. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST).* 1–2. https://doi.org/10.1109/ICST.2015.7102637

[4] Cesare Alippi. 2014. *Intelligence for Embedded Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg.

[5] Christel Baier, Joost-Pieter Katoen, et al. 2008. *Principles of model checking.* MIT press Cambridge.

[6] A. Banerjee, K. K. Venkatasubramanian, T. Mukherjee, and S. K. S. Gupta. 2012. Ensuring Safety, Security, and Sustainability of Mission-Critical Cyber-Physical Systems. *Proc. IEEE* 100, 1 (Jan 2012), 283–299. https://doi.org/10.1109/JPROC.2011.2165689

[7] M. M. Bersani and M. García-Valls. 2016. The Cost of Formal Verification in Adaptive CPS. An Example of a Virtualized Server Node. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE).* 39–46. https://doi.org/10.1109/HASE.2016.46

[8] Keith Beven and Andrew Binley. 1992. The future of distributed models: Model calibration and uncertainty prediction. *Hydrological Processes* 6, 3 (1992), 279–298. https://doi.org/10.1002/hyp.3360060305

[9] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. 2012. Viewpoints, Formalisms, Languages, and Tools for Cyber-physical Systems. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling (MPM '12).* ACM, New York, NY, USA, 49–54. https://doi.org/10.1145/2508443.2508452

[10] Betty H.C. Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *MODELS.* LNCS, Vol. 5795. Springer Berlin Heidelberg, 468–483. https://doi.org/10.1007/978-3-642-04425-0_36

[11] Edmund M. Clarke and Paolo Zuliani. 2011. Statistical Model Checking for Cyber-physical Systems. In *Proceedings of the 9th International Conference on Automated Technology for Verification and Analysis (ATVA'11).* Springer-Verlag, Berlin, Heidelberg, 1–12. http://dl.acm.org/citation.cfm?id=2050917.2050919

[12] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS 2008, Held as Part of ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, C. R. Ramakrishnan and Jakob Rehof (Eds.). 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[13] Sebastian Elbaum and David S. Rosenblum. 2014. Known Unknowns: Testing in the Presence of Uncertainty. In *Proc. FSE '14.* Hong Kong, China, 833–836. https://doi.org/10.1145/2635868.2666608

[14] Marisol García-Valls and Roberto Baldoni. 2015. Adaptive Middleware Design for CPS: Considerations on the OS, Resource Managers, and the Network Run-time. In *Proceedings of the 14th International Workshop on Adaptive and*

Reflective Middleware (ARM 2015). ACM, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/2834965.2834968

[15] Marisol García-Valls, Christian Calva-Urrego, Juan A. de la Puente, and Alejandro Alonso. 2017. Adjusting middleware knobs to assess scalability limits of distributed cyber-physical systems. *Computer Standards & Interfaces* 51 (2017), 95 – 103. https://doi.org/10.1016/j.csi.2016.11.003

[16] Marisol García-Valls, Diego Perez-Palacin, and Raffaela Mirandola. 2014. Time-Sensitive Adaptation in CPS Through Run-Time Configuration Generation and Verification. In *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC '14)*. IEEE Computer Society, 332–337. https://doi.org/10.1109/COMPSAC.2014.55

[17] Andrew Gelman, B. Carlin John, S. Stern Hal, and Donald B. Rubin. 2014. *Bayesian data analysis*. Chapman Hall CRC.

[18] Herman Geuvers, Adam Koprowski, Dan Synek, and Eelis van der Weegen. 2010. Automated Machine-Checked Hybrid System Safety Proofs. In *Interactive Theorem Proving: First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, Matt Kaufmann and Lawrence C. Paulson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 259–274. https://doi.org/10.1007/978-3-642-14052-5_19

[19] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[20] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. 2012. Search Based Software Engineering: Techniques, Taxonomy, Tutorial. In *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*, Bertrand Meyer and Martin Nordio (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–59. https://doi.org/10.1007/978-3-642-25231-0_1

[21] J. He, Y. Geng, Y. Wan, S. Li, and K. Pahlavan. 2013. A Cyber Physical Test-Bed for Virtualization of RF Access Environment for Body Sensor Network. *IEEE Sensors Journal* 13, 10 (Oct 2013), 3826–3836. https://doi.org/10.1109/JSEN.2013.2271721

[22] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. 1995. HYTECH: the next generation. In *Proceedings 16th IEEE Real-Time Systems Symposium*. 56–65. https://doi.org/10.1109/REAL.1995.495196

[23] John H Holland. 1975. *Adaptation in natural and artificial systems*. U Michigan Press.

[24] Huang-Ming Huang, Terry Tidwell, Christopher Gill, Chenyang Lu, Xiuyu Gao, and Shirley Dyke. 2010. Cyber-physical Systems for Real-time Hybrid Structural Testing: A Case Study. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS '10)*. ACM, New York, NY, USA, 69–78. https://doi.org/10.1145/1795194.1795205

[25] Chii-Ruey Hwang. 1988. Simulated annealing: Theory and applications. *Acta Applicandae Mathematica* 12, 1 (1988), 108–111. https://doi.org/10.1007/BF00047572

[26] Aaron Kane, Thomas Fuhrman, and Philip Koopman. 2014. Monitor Based Oracles for Cyber-Physical System Testing: Practical Experience Report. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '14)*. IEEE Computer Society, Washington, DC, USA, 148–155. https://doi.org/10.1109/DSN.2014.28

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671 arXiv:http://science.sciencemag.org/content/220/4598/671.full.pdf

[28] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele. 2012. A hybrid approach to cyber-physical systems verification. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. 688–696. https://doi.org/10.1145/2228360.2228484

[29] L. S. Lasdon, A. D. Waren, A. Jain, and M. Ratner. 1978. Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming. *ACM Trans. Math. Softw.* 4, 1 (March 1978), 34–50. https://doi.org/10.1145/355769.355773

[30] E. A. Lee. 2008. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 363–369. https://doi.org/10.1109/ISORC.2008.25

[31] Jian Lü, Yu Huang, Chang Xu, and Xiaoxing Ma. 2013. Theories of Programming and Formal Methods. Springer-Verlag, Berlin, Heidelberg, Chapter Managing Environment and Adaptation Risks for the Internetware Paradigm, 271–284. http://dl.acm.org/citation.cfm?id=2554641.2554658

[32] Jian Lü, Xiaoxing Ma, Yu Huang, Chun Cao, and Feng Xu. 2009. Internetware: A Shift of Software Paradigm. In *Proceedings of the First Asia-Pacific Symposium on Internetware (Internetware '09)*. ACM, New York, NY, USA, Article 7, 9 pages. https://doi.org/10.1145/1640206.1640213

[33] Atif Mashkoor and Osman Hasan. 2012. *Formal Probabilistic Analysis of Cyber-Physical Transportation Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 419–434. https://doi.org/10.1007/978-3-642-31137-6_32

[34] MathWorks. 2017. MATLAB. http://www.mathworks.com/. (2017).

[35] H. Mei, G. Huang, and T. Xie. 2012. Internetware: A Software Paradigm for Internet Computing. *Computer* 45, 6 (June 2012), 26–31. https://doi.org/10.1109/MC.2012.189

[36] S. Mitra, T. Wongpiromsarn, and R. M. Murray. 2013. Verifying Cyber-Physical Interactions in Safety-Critical Systems. *IEEE Security Privacy* 11, 4 (July 2013), 28–37. https://doi.org/10.1109/MSP.2013.77

[37] Jorge J. Moré and D. C. Sorensen. 1983. Computing a Trust Region Step. *SIAM J. Sci. Statist. Comput.* 4, 3 (1983), 553–572. https://doi.org/10.1137/0904038

[38] John Neter, Michael H Kutner, Christopher J Nachtsheim, and William Wasserman. 1996. *Applied linear statistical models*. Vol. 4. Irwin Chicago.

[39] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-physical Systems: The Next Computing Revolution. In *Proceedings of the 47th Design Automation Conference (DAC '10)*. ACM, New York, NY, USA, 731–736. https://doi.org/10.1145/1837274.1837461

[40] Singiresu S Rao and SS Rao. 2009. *Engineering optimization: theory and practice*. John Wiley & Sons.

[41] D. Ricketts, G. Malecha, M. M. Alvarez, V. Gowda, and S. Lerner. 2015. Towards verification of hybrid systems in a foundational proof assistant. In *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*. 248–257. https://doi.org/10.1109/MEMCOD.2015.7340492

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. MIT Press, Cambridge, MA, USA, Chapter Learning Internal Representations by Error Propagation, 318–362. http://dl.acm.org/citation.cfm?id=104279.104293

[43] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. 2003. *Artificial intelligence: a modern approach*. Vol. 2. Prentice hall Upper Saddle River.

[44] Muhammad Usman Sanwal and Osman Hasan. 2013. *Formal Verification of Cyber-Physical Systems: Coping with Continuous Elements*. Springer Berlin Heidelberg, Berlin, Heidelberg, 358–371. https://doi.org/10.1007/978-3-642-39637-3_29

[45] Mary C. Seiler and Fritz A. Seiler. 1989. Numerical Recipes in C: The Art of Scientific Computing. *Risk Analysis* 9, 3 (1989), 415–416. https://doi.org/10.1111/j.1539-6924.1989.tb01007.x

[46] Joseph Sifakis and Sergio Yovine. 1996. Compositional specification of timed systems. In *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 345–359.

[47] Lenardo C. Silva, Mirko Perkusich, Frederico M. Bublitz, Hyggo O. Almeida, and Angelo Perkusich. 2014. A Model-based Architecture for Testing Medical Cyber-physical Systems. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14)*. ACM, New York, NY, USA, 25–30. https://doi.org/10.1145/2554850.2555028

[48] J.A.K. Suykens and J. Vandewalle. 1999. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters* 9, 3 (1999), 293–300. https://doi.org/10.1023/A:1018628609742

[49] P. Tabuada, S. Y. Caliskan, M. Rungger, and R. Majumdar. 2014. Towards Robustness for Cyber-Physical Systems. *IEEE Trans. Automat. Control* 59, 12 (Dec 2014), 3151–3163. https://doi.org/10.1109/TAC.2014.2351632

[50] T. Tidwell, X. Gao, H. M. Huang, C. Lu, S. Dyke, and C. Gill. 2009. Towards Configurable Real-Time Hybrid Structural Testing: A Cyber-Physical System Approach. In *2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. 37–44. https://doi.org/10.1109/ISORC.2009.41

[51] Tazio Vanni, Jonathan Karnon, Jason Madan, Richard G White, W John Edmunds, Anna M Foss, and Rosa Legood. 2011. Calibrating Models in Economic Evaluation. *PharmacoEconomics* 29 (Jan. 2011), 35–49.

[52] Allan J Volponi. 1994. Sensor error compensation in engine performance diagnostics. In *ASME 1994 International Gas Turbine and Aeroengine Congress and Exposition*. American Society of Mechanical Engineers, V005T15A008.

[53] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and Computing* 4, 2 (1994), 65–85. https://doi.org/10.1007/BF00175354

[54] Fang-Jing Wu, Yu-Fen Kao, and Yu-Chee Tseng. 2011. From wireless sensor networks towards cyber physical systems. *Pervasive and Mobile Computing* 7, 4 (2011), 397 – 413. https://doi.org/10.1016/j.pmcj.2011.03.003

[55] Chang Xu, Wenhua Yang, Xiaoxing Ma, Chun Cao, and Jian Lu. 2013. Environment rematching: Toward dependability improvement for self-adaptive applications. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE '13)*. 592–597. https://doi.org/10.1109/ASE.2013.6693118

[56] Wenhua Yang, Chang Xu, Yepang Liu, Chun Cao, Xiaoxing Ma, and Jian Lu. 2014. Verifying Self-adaptive Applications Suffering Uncertainty. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 199–210. https://doi.org/10.1145/2642937.2642999

[57] Lichen Zhang, Jifeng He, and Wensheng Yu. 2013. Test Case Generation from Formal Models of Cyber Physical System. *International Journal of Hybrid Information Technology* 6, 3 (2013), 15–24.

[58] Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. 2016. Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model. In *Modelling Foundations and Applications: 12th European Conference, ECMFA 2016, Held as Part of STAF 2016, Vienna, Austria, July 6-7, 2016, Proceedings*, Andrzej Wąsowski and Henrik Lönn (Eds.). Springer International Publishing, Cham, 247–264. https://doi.org/10.1007/978-3-319-42061-5_16

[59] X. Zheng. 2014. Physically informed assertions for cyber physical systems development and debugging. In *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*. 181–183. https://doi.org/10.1109/PerComW.2014.6815195

Received September 2016; revised March 2017; accepted May 2017