Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2018-IJ-001

2018-IJ-001

Systematically Ensuring The Confidence of Real Time Home Automation IoT Systems

Lei Bu, Wen Xiong, Chieh-Jan Mike Liang, Shi Han, Dongmei Zhang, Shan Lin, Xuandong Li

ACM Transactions on Cyber-Physical Systems 2018

Systematically Ensuring the Confidence of Real-Time Home Automation IoT Systems

LEI BU and WEN XIONG, Nanjing University
CHIEH-JAN MIKE LIANG, SHI HAN, and DONGMEI ZHANG, Microsoft Research
SHAN LIN, Stony Brook University
XUANDONG LI, Nanjing University

Recent advances and industry standards in Internet of Things (IoT) have accelerated the real-world adoption of connected devices. To manage this hybrid system of digital real-time devices and analog environments, the industry has pushed several popular home automation IoT (HA-IoT) frameworks, such as If-This-Then-That (IFTTT), Apple HomeKit, and Google Brillo. Typically, users author device interactions by specifying the triggering sensor event and the triggered device command. In this seemingly simple software system, two dominant factors govern the system confidence properties with respect to the physical world. First, IoT users are largely nonexperts who lack the comprehensive consideration regarding potential impact and joint effect with existing rules. Second, while the increasing complexity of IoT devices enables fine-grained control (e.g., heater temperature) of continuous real-time environments, even two simply connected devices can have a huge state space to explore. In fact, bugs that wrongfully control devices and home appliances can have ramifications on system correctness and even user physical safety. It is crucial to help users to make sure the system they created meets their expectation. In this article we introduce how techniques from hybrid automata can be practically applied to assist nonexpert IoT users in the confidence checking of such hybrid HA-IoT systems. We propose an automated framework for end-to-end programming assistance. We build and check the Linear Hybrid Automata (LHA) model of the system automatically. We also present a quantifier elimination-based method to analyze the counterexample found and synthesize fix suggestions. We implemented a platform, MenShen, based on this framework and proposed techniques. We conducted sets of real HA-IoT case studies with up to 46 devices and 65 rules. Empirical results show that MenShen can find violations and generate rule fix suggestions in only 10 seconds.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Software and its engineering → Model checking**;

Additional Key Words and Phrases: IFTTT, home automation, internet of things, linear hybrid automata, automatic modeling and verification, fix suggestion

This artilce is supported in part by the National Natural Science Foundation of China (No.61690204, No.61632015, No.61561146394, No.61572249, No.61472179), in which No.61561146394 is a Joint NSFC-ISF Research Program, jointly funded by the National Natural Science Foundation of China and the Israel Science Foundation. Lei Bu is also supported by the Microsoft Research Asia Collaborative Research Program. Shan Lin is supported by the NSF CNS 1553273 and CNS 1463722.

Authors' addresses: L. Bu, W. Xiong, and X. Li are with the Computer Science Department, Xianlin Campus, Nanjing University, Nanjing, Jiangsu, China, 210023; emails: {bulei, xiongwen, lxd}@nju.edu.cn; C.-J. M. Liang, S. Han, and D. Zhang, Building 2, No. 5 Dan Ling Street, Haidian District, Beijing, China, 100080; emails: {liang.mike, shihan, dongmeiz}@microsoft.com; S. Lin is with Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794-2350; email: shan.x.lin@stonybrook.edu.

https://doi.org/10.1145/3185501

ACM Reference format:
Lei Bu, Wen Xiong, Chieh-Jan Mike Liang, Shi Han, Dongmei Zhang, Shan Lin, and Xuandong Li. 2018. Systematically Ensuring the Confidence of Real-Time Home Automation IoT Systems. *ACM Trans. Cyber-Phys. Syst.* 2, 3, Article 22 (June 2018), 23 pages.
https://doi.org/10.1145/3185501

1 INTRODUCTION

With the rapid advancement of computing technology, the computing paradigm of Cyber-Physical Systems (CPS) (Lee 2006) has emerged in the past decade. Under this paradigm, sensor data can be acquired and processed in real time, which then drives intelligence (Lin et al. 2008). Furthermore, riding on the momentum of the Internet of Things (IoT), the industry has pushed for standards that enable more connected devices to interoperate. However, central to this connected vision is an IoT control framework, which manages the hybrid system of digital real-time devices and analog environments in a space. Many home owners have benefited from this IoT control framework. For example, they can author automation tasks that send SMS messages to the ho homeowner when the kitchen has a high smoke level.

The industry has pushed several offerings in home automation IoT (HA-IoT) services, such as If-This-Then-That (IFTTT.com) (ift 2011), Apple HomeKit (ah 2016), and Google Brillo (gb 2016). Interestingly, the user base of home automation largely consists of nonexperts who have insufficient background with programming hybrid control systems. Therefore, these industry offerings simplify authoring an automation task down to authoring a set of intuitive event-triggered rules – or IFTTT-style rules. IFTTT rules are popular among HA-IoT services, and one HA-IoT service, IFTTT.com, has tens of thousands of active users and more than 340,000 shared rules. An IFTTT-style rule is in the format of `if A then do B`, where `A` is a triggering sensor event and `B` is a triggered device command. We illustrate the automation task with an example of two rules that regulate the CO concentration in a given space to be under 200 ppm.[1]

```
IF Smart_Fan.CO_reading == 195 THEN execute Alarm.TURN_ON command
IF Alarm.TURN_ON.Signal ==TRUE THEN execute Smart_Fan.ACTIVATE command
```

This CO example has a clear relation to user safety. There is also another widely used rule which concerns keeping room temperature within certain range and also concerns energy saving.[2]

```
IF Room. Temperature_reading ==20 THEN execute HVAC.TURN_OFF command
IF Room. Temperature_reading ==28 THEN execute HVAC.TURN_ON command
IF HVAC.TURN_ON.Signal == TRUE THEN execute Window.CLOSE command
```

However, while individual IFTTT-style rules are simple to author, reasoning about their confidence (i.e., whether a system's real-time behavior conforms to a user's expectation) is a complicated task with implications for system confidence. This is crucial as wrongfully actuating IoT devices and appliances could have ramifications for user physical well-being (e.g., high CO concentration). Such reasoning needs to accurately model the behavior of a system of devices over the time domain. Challenges arise from the fact that (1) environment and system variables are

[1] We note that this CO concentration case is inspired by a rule uploaded by normal users on IFTTT.com, https://ifttt.com/recipes/368595-turn-on-your-air-purifier-when-the-air-quality-decreases.
[2] This HVAC case is also inspired by a rule on IFTTT.com, https://ifttt.com/recipes/182684-if-the-temperature-inside-drops-below-degrees-then-turn-off-a-c.

changing continuously, (2) events can happen at any time, and (3) the number of interactions to inspect increases with the number of automation rules. More importantly, while typical real- time control systems are maintained by domain experts, HA-IoT systems are operated by nonexpert homeowners. Therefore, building on formal model checking, we investigate *new approaches and tools to help non-expert IoT users in systematically realizing high-confidence real-time HA-IoT systems.*

Our system design is guided by two main principles. First, there is an increasing number of IoT appliances that deal with *real-time continuous environments* with *dynamic laws* and *time delays.* In our CO example, if the smart fan is activated, it can decrease CO concentrations according to a dynamic law (ODE $dCO/dt = -2$). So, the 195 ppm threshold of the purifier seems to be correct in that the CO level will not go over 200 ppm. However, due to the existence of a time delay in rule reaction, the CO level could exceed the threshold before the smart fan is activated. The HVAC example shares a similar story as well. Related efforts either simplify the problem down to discrete values (Liang et al. 2015, 2016), or they do not support arbitrary continuous behavior (Croft et al. 2015), which may cause false negatives in the verification. Second, all steps of system modeling, specification verification, and violation fixing should be as transparent as possible to the user. Unlike typical CPS systems with domain experts (Clarke et al. 2008), relying on nonexpert users is impractical due to their lack of background knowledge.

This article realizes an end-to-end programming assistance system to automate the modeling, checking, and fixing of HA-IoT systems. Specifically, this article makes the following contributions:

Hybrid Automata Model Checking of Real-Time HA-IoT System. As IoT devices are becoming a customized commodity, it is important to help IoT users to ensure the confidence of the automation systems they build. Motivated by IoT-specific characteristics, we take the first step to fill this gap with hybrid automata model checking. First, by using hybrid automata, it is possible for us to model and check the complex arbitrary continuous real-time behavior of an analog system. This was previously unachievable through related efforts (Croft et al. 2015). Second, in contrast to efforts which require user intervention in modeling and debugging (Liang et al. 2015), our work tries to automate all stages for nonexpert IoT users.

Counterexample-guided Fix Suggestions. With each specification violation, the common practice is for the verification tool to output a counterexample, which is then analyzed by domain experts to fix the software problem. However, such an assumption does not hold in the case of nonexpert IoT users. Instead, what should be provided is root cause analysis that pinpoints the IFTTT rule to fix. To this end, we present a method to automatically parameterize the system and synthesize specific parameter values through the Quantifier Elimination (QE) (Monniaux 2008) technique.

System Implementation and Real Case Evaluation. We have been operating an IoT programing platform, MenShen, which implements our techniques. We check and fix a large number of real cases, including 46 devices and 65 rules, in only 7.5 seconds. In MenShen, we also support and appreciate interaction with users. If users can select the specific rules they want to fix, or give a preferred range of rule parameters, MenShen can finish the task in only 4 seconds with high user acceptance.

Structure of the Article. The IFTTT-style IoT programming platform studied in this article is presented in the next section. Section 2 also recaps the definition and verification technique of LHA, which works as the underlying decision procedure of this article. After that, we present the architecture of our modeling, verification, and repair framework in Section 3. The details of the automatic modeling technique and the verifiable schema behind this is introduced in Section 4.

The verification technique and our new proposed QE-based fix suggestion is presented in Section 5. System implementation and evaluation is reported in Section 6. Sections 7 and 8 briefly discuss the limitation of this work and review related work. Finally, the conclusion is stated in Section 9.

2 BACKGROUND

2.1 IFTTT-Style Programming Paradigm for HA-IoT

Supporting interactions among digital real-time devices and analog environments, a software system is at the center of home automation IoT (HA-IoT). This section discusses the current state of HA-IoT software systems in terms of the programming paradigm and unfilled gaps.

Similar to wireless sensor networks, HA-IoT is a software system that reacts to changes in analog environments. For example, if room temperature is below a threshold, the heater should be turned on. As such, the event-driven programming paradigm is widely used by real-world HA-IoT software systems such as IFTTT (ift 2011), Apple HomeKit (ah 2016), and Google Brillo (gb 2016). Popularized by IFTTT, this paradigm is referred to as *IFTTT-style programming*. In IFTTT-style programming, an automation program consists of IFTTT rules, and these rules are executed in parallel. Individual rules follow the format of `if A then do B`, where `A` is a triggering sensor event and `B` is a triggered device command. In the preceding example the former is the room temperature, and the latter is turning on the heater. By crawling the IFTTT website, we found that more than 34,000 HA-IoT rules have been created and shared by average users, including alerting when room CO level is too high, turning off the heater when no one is home, and the like. Depending on the number and types of connected devices deployed, a normal scale real case HA-IoT deployment typically has 10 to 30 IFTTT rules.

We argue that current HA-IoT software systems have the following unfilled gaps.

Lack of Automatic HA-IoT Confidence Verification. Since HA-IoT is a software system that controls IoT-enabled devices and appliances, any software bug (or unexpected behavior) can have ramifications in the real world and even risk user safety. For example, in our previous CO example, if the smart fan starts too late, the room CO level can go beyond 200 ppm and be harmful. Therefore, we argue that automatic confidence verification of certain systems is crucial.

For formal modeling and verification to be practical, the HA-IoT programming tool suite must be able to automate as much as possible. While HA-IoT software systems for typical houses might not be as complicated as expert-built real-time control systems (such as for trains), the state space behind the IFTTT-style rules can quickly grow beyond nonexpert users' comprehension. Furthermore, the time delay and different dynamic laws in the system are already too difficult for average end-users to understand. As we discuss in the next sections, techniques from hybrid automata can be the foundation of a confidence verification solution for HA-IoT solutions from the industry.

Lack of Debugging Feedback for Nonexpert IoT Users. When a specification violation is identified, the common practice is for the verification tool to provide domain experts with a counterexample or a sequence of system state transitions leading to the violation. Unfortunately, this feedback lacks sufficient information for nonexpert users to comprehend the violation and pin-point the rules to fix. First, since a violation is caused by a sequence of rule execution, there is no clear indication which rule is at fault. And the overhead of understanding the effect of each rule on the output can be high. Second, changing a rule in the execution sequence does not necessarily fix the problem, as doing so may also change which rules are triggered later in the sequence.

To fill this gap of debugging assistance, we propose using QE (Monniaux 2008) as the foundation to provide actionable feedback.

2.2 Modeling and Verification of Hybrid System

As we can see from the CO example and the previous subsection, the behavior of a HA-IoT system is tangling with both discrete logic control and continuous time behavior. Such a system is called a *hybrid system*. Linear Hybrid Automata (LHA) is a class of widely used formal languages for modeling hybrid systems (Henzinger 1996). The model checking (Clarke et al. 2001) problem for LHA is considerably difficult, and the reachability checking problem is undecidable (Henzinger et al. 1998). Classical techniques try to compute the whole reachable state space of the LHA by using the expensive polyhedral computation, which is sensitive to continuous variables and not guaranteed to terminate.

Recently, Bounded Model Checking (BMC) (Biere et al. 2003; Audemard et al. 2005) has attracted lots of attention as an alternative to general model checking. The basic idea is to look for a counterexample in the given bound threshold instead of the complete state space. In this manner, the state space needed to search is controlled and thus can be efficiently checked.

Now, let's give a brief introduction to the definition of LHA (Henzinger 1996) and a state-of-the-art BMC reachability checking technique for LHA, the *path-oriented bounded reachability analysis* (Li et al. 2007; Bu and Li 2011).

Definition 2.1. An LHA H is a tuple $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$, where

— X is a finite set of real-valued variables; Σ is a finite set of event labels.
— V is a finite set of *locations*; v_I is the *initial location*.
— E is a *transition relation* whose elements are of the form $(v, \sigma, \phi, \psi, v')$, where $v, v' \in V$, $\sigma \in \Sigma$, ϕ is a set of *transition guards* of the form $a \leq \sum_{i=0}^{l} c_i x_i \leq b$, and ψ is a set of *reset actions* of the form $x := c$ where $x_i \in X$, $x \in X$, a, b, c , and c_i are real numbers (a, b may be ∞).
— α is a labeling function which maps each location in $V - \{v_I\}$ to a *location invariant* which is a set of *variable constraints* of the form $a \leq \sum_{i=0}^{l} c_i x_i \leq b$, where $x_i \in X$, a, b and c_i are real numbers (a, b may be ∞).
— β is a labeling function which maps each location in $V - \{v_I\}$ to a set of *flow conditions*, which are of the form $\dot{x} \in [a, b]$, where $x \in X$, and a, b are real numbers ($a \leq b$). For any location v, for any $x \in X$, there is one and only one flow condition $\dot{x} \in [a, b] \in \beta(v)$.

For a group of LHA $\{H_1, H_2, \ldots, H_n\}$, their composition, denoted as $N = H_1||H_2|| \ldots ||H_m$, is defined as a LHA from synchronizing all components with respect to the same event labels. Labels shared by several LHA models are called *Shared Labels*. The semantic of the shared label-guided synchronization is simple. Suppose several LHA models have a shared label; firing this shared label triggers the same transition in all models at the same time.

Definition 2.2. For an LHA $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$, a *reachability specification*, denoted as $\mathcal{R}(v, \varphi)$, consists of a location v in H and a set φ of variable constraints of the form $a \leq \sum_{i=0}^{l} c_i x_i \leq b$ where $x_i \in X$ for any i ($0 \leq i \leq l$), a, b , and c_i ($0 \leq i \leq l$) are real numbers.

We use the sequences of locations to represent the evolution of an LHA from location to location. For an LHA $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$, a *path segment* is a sequence of locations of the form $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0, \psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1, \psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1}, \psi_{n-1})} \langle v_n \rangle$, which satisfies $(v_i, \sigma_i, \phi_i, \psi_i, v_{i+1}) \in E$ for each i ($0 \leq i < n$). A *path* in H is a path segment starting from the initial location v_I.

The question of whether a given path ρ in an LHA model H satisfies specification $\mathcal{R}(v, \varphi)$ has been well studied in Li et al. (2007) and Bu and Li (2011). The basic idea is to describe the state space of ρ by encoding all the semantical elements, including transition guards, resets, location invariants, and flow conditions along this path into a formula together, denoted as Ψ. Then,

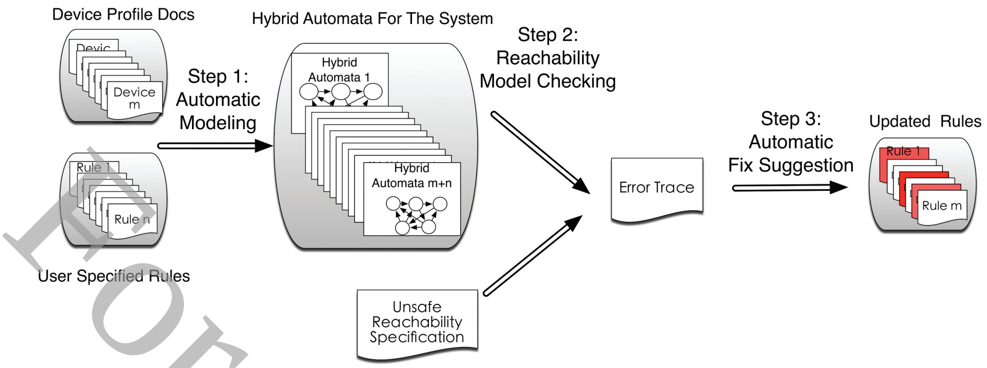

Device Profile Docs
Devic
Device m
Rule 1
Rule n
User Specified Rules
Step 1: Automatic Modeling
Hybrid Automata For The System
Hybrid Automata 1
Hybrid Automata m+n
Step 2: Reachability Model Checking
Unsafe Reachability Specification
Error Trace
Step 3: Automatic Fix Suggestion
Updated Rules
Rule 1
Rule m

Fig. 1. The architecture of the framework.

whether ρ satisfies $\mathcal{R}(v, \varphi)$ can be translated into the problem of the feasibility of Ψ, which can be solved efficiently by linear programming.[3]

As we all know, the basic idea of BMC is to search for a counterexample in executions whose length is bounded by some integer k. Given an LHA, the number of candidate paths with length no longer than k is finite. Therefore, if we enumerate and check all the paths in the bound one by one, the BMC problem can be tackled. This is known as *path-oriented bounded analysis* (Bu and Li 2011; Xie et al. 2014) of LHA.

Now we've reviewed the status of the latest HA-IoT industry offerings, the definition of LHA, and also the path-oriented BMC analysis technique for LHA. In the next section, we'll show how we can use such techniques in the HA-IoT industry.

3 FRAMEWORK OVERVIEW

As summarized in the preceding sections, the IFTTT-style HA-IoT system gives users high autonomy to build their own customized smart home automation system, albeit with potential confidence risks. As HA-IoT systems have extremely close relations with users' daily life, it is crucial to offer a mechanism which can help users increase the confidence of certain systems. In this article, we propose an *automated framework for end-to-end programming assistance* to tackle this problem by performing the modeling, checking, and fixing of such systems automatically. The framework shown in Figure 1 consists of the following parts:

- **Linear Hybrid Automata Automatic Modeling:** The first phase of our framework is to generate the LHA models according to the devices and IFTTT rules in the system. It is unrealistic to ask an average end-user to build the models of the system. Therefore, we design a specific schema where device manufacturers can present necessary device information according to the format of such a schema. Then, we can automatically build the LHA model of the system from device documentation and the IFTTT rules.
- **Reachability Analysis of LHA Model:** The second phase of our framework is to check the user-specified unwanted reachability specification by the path-oriented BMC checking procedure mentioned in Section 2. Clearly, if the unwanted target is not reachable, the system is good. Otherwise, we'll get an error trace, which describes the sequence of system events leading to a "bad" state.

[3] Due to space limitations, we give a short review of the encoding of Ψ in the appendix. Readers are also referred to (Li et al. 2007; Bu and Li 2011) for details of such techniques.

- **Counterexample-guided Fix Suggestion Synthesis:** If the model fails verification, the third phase is to help the user in debugging the counterexample. Again, as typical IoT users have insufficient knowledge in software testing and verification, providing guidance (e.g., fix suggestions) can be very helpful. To do so, we propose a method that first parameterizes the system and subsequently solves the free parameters using QE techniques.

4 VERIFIABLE SCHEMA AND AUTOMATIC LHA MODELING

In this section, we introduce the first part of our framework: Automatic LHA modeling. Clearly, it is impractical and unreasonable to ask the average user to build a model for her devices and rules manually. Therefore, such a task should be conducted automatically and systematically, if possible.

As we can see from the CO example, the aspects affecting the behavior of the HA-IoT system include high-level control logic, the time delay and dynamic laws inherent in the system, the synchronization among devices and rules, and so on. Therefore, LHA, which is the simplest model that can address all these aspects, is the most suitable modeling language for such HA-IoT systems. Now the question is: How can we build the LHA model for such a system automatically?

4.1 Verifiable Device Schema

Before building an LHA model, we need to get all the information needed about the device. Actually, IoT-enabled devices typically have a presence-advertising feature. It is common to see manufacturers list the working modes of the device, publicly observable variables, executable APIs, and even simple dynamic laws in different places (e.g., advertisements, user manuals, device websites), piece by piece.

Clearly, manufacturers know all the information about the device. For the sake of automatic model generation, we argue each device should come with a profile documentation organized in a specific format that can express all the necessary information about the device. Actually, the industry has proposed such a standard to present device information in an organized way, such as the Device Registry for AWS IoT (DR) (aws 2015). Similar to the style of DR, which is readable and writable by manufacturers, we give the format of the verifiable device schema as follows:

- Device *Type* and *SN*, which indicate the type and the serial number (SN) of the device.
- A set of *System Variables*, which indicates the *environmental variables* affected by the device or the *internal data* kept by the device. In detail, if the value of a variable is observable by the user, we mark such variable as *public*.
- A set of device *Working Modes*, which represents the high-level discrete working modes of the device. In detail, for each mode, descriptions should include *dynamic* information on how the system variables will evolve in such mode. For example, in our CO example, when the smart fan is activated, the CO concentration in the room is going down by $dCO/dt = -2$.
- A set of *Transitions*, which indicates the internal mode-changing logic of the device, such as under which *triggering condition* the device will change from mode A to mode B. There is also a special boolean flag, *Signal*. If this flag is true, it means the execution of the transition is a *triggering condition event* that the environment can observe.
- Finally, a set of *APIs*, which describes the kind of *triggered commands* that can be called by users and other devices. It has the same structure as *Transitions*. The only difference is that only the API can appear in the right-hand side of an IFTTT rule.

JSON (jso 2009) is a widely used data format that can be parsed efficiently. We design a specific file format to express the preceding information in a JSON file. For example, Listing 1 and Listing 2 in Figure 2 give the JSON files for the devices in the CO example.

Listing 1. Smart Fan

```
{
  ”Device”:
  {
    ”Type”:”Smart Fan”,
    ”SN”:”0001”,
    ”InternalVari”:
    [
      {
        ”Name”:”CO_level”,
        ”Type”:”double”,
        ”Default”:”20”,
        ”public”:”true”
      }
    ],
    ”WorkingMode”:
    [
      {
        ”Name”:”Closed”,
        ”Dynamic”:
        [
          {
            ”VariableName”:”CO_level”,
            ”ChangeRate”:”[−1,1]”
          }
        ],
        ”Invariant”:”true”,
        ”init”:”true”,
      },
      {
        ”Name”:”Working”,
        ”Dynamic”:
        [
          {
            ”VariableName”:”CO_level”,
            ”ChangeRate”:”−2”
          }
        ],
        ”Invariant”:”true”,
        ”init”:”false”,
      }
    ],
    ”Transitions”:
    [

    ],
    ”API”:[
      {
        ”Name”:”ACTIVATE”,
        ”StartMode”:”Closed”,
        ”EndMode”:”Working”,
        ”Trigger”:””,
        ”Assignments”:[ ],
        ”Signal”:”true”
      },
      {
        ”Name”:”PAUSE”,
        ”StartMode”:”Working”,
        ”EndMode”:”Closed”,
        ”Trigger”:””,
        ”Assignments”:[ ],
        ”Signal”:”true”
      }
    ]
  }
}
```

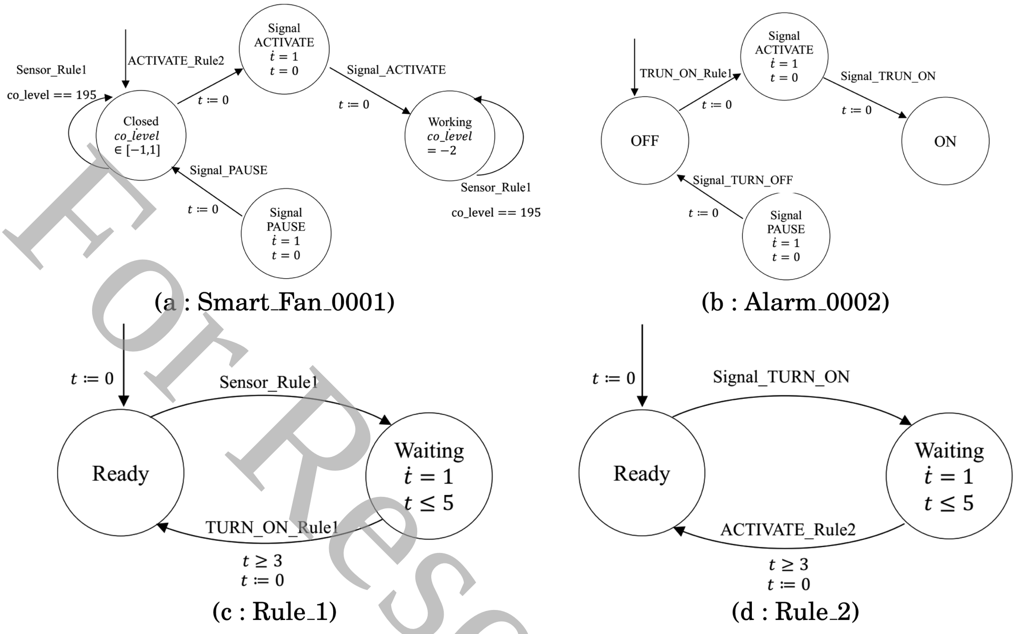
Listing 2. Alarm

```
{
  ”Device”:
  {
    ”Type”:”Alarm”,
    ”SN”:”0002”,
    ”InternalVari”:
    [

    ],
    ”WorkingMode”:
    [
      {
        ”Name”:”ON”,
        ”Dynamic”:
        [
        ],
        ”Invariant”:”true”,
        ”init”:”false”,
      },
      {
        ”Name”:”OFF”,
        ”Dynamic”:
        [
        ],
        ”Invariant”:”true”,
        ”init”:”true”,
      }
    ],
    ”Transitions”:
    [

    ],
    ”API”:[
      {
        ”Name”:”TURN_ON”,
        ”StartMode”:”OFF”,
        ”EndMode”:”ON”,
        ”Trigger”:””,
        ”Assignments”:[ ],
        ”Signal”:”true”
      },
      {
        ”Name”:”TURN_OFF”,
        ”StartMode”:”ON”,
        ”EndMode”:”OFF”,
        ”Trigger”:””,
        ”Assignments”:[ ],
        ”Signal”:”true”
      }
    ]
  }
}
```

Fig. 2. Json docs for the CO example.



(a : Smart_Fan_0001) (b : Alarm_0002)

(c : Rule_1) (d : Rule_2)

Fig. 3. LHA models generated for the CO example.

4.2 From Schema To LHA

Now, we present how to automatically build the LHA model from the device schema.

First, the LHA model for each device is modeled in the following way:

- The name, variables, locations, and flow conditions of the LHA model can be generated directly based on the device schema.
- Each transition in the *Transitions* section becomes a transition in the LHA model.
    - If the *Signal* flag of the transition is true, then mark the label of the corresponding transition as *Signal_Transition.Name*. In this manner, other LHA models can communicate with this model using this shared label.
- The *API* section is treated differently. In detail:
    - If the *Signal* flag of the transition is false, then we treat it as a normal transition first. Then, as the API could be called by other components/rules, whenever there is a new caller, we will add a new transition with the label *API.Name_Caller.Name.*[4]
    - If the *Signal* flag is true, we add an intermediate location between the source and target location of a certain API call. More specifically, we force the dwelling time of this intermediate location to be 0. Then, the previous API transitions will point to this intermediate location, and a new transition will be added from this intermediate location to the original target location with the label *Signal_API.Name*. In this manner, no matter which caller executes this API, other components can receive the same signal. For example, we can see such locations and transitions in Figure 3(a) and (b).

[4] We must distinguish different callers by transitions with different labels, because if different callers use the same label for one API, then they will be forced to fire the transition at the same time according to the synchronization semantic of LHA.

For IFTTT rules, as each rule *i* has the same structure, the modeling can be done in a structural way.

- The structure of the IFTTT rule automaton is simple. There are two locations, where transitions wait for the enabling of the triggering condition and are ready to execute the triggered command.
- As the *triggered command* part of a rule can only be an API of a device, the label of the corresponding transition is *API.Name_Caller.Name*. In this case, it can communicate with the device automaton by this shared label.
- The *triggering condition* part could be the occurrence of an event/signal or a triggering condition expression ϕ of an observable variable
  - If it is a signal, the label of the corresponding transition is *Signal_Transition.Name*.
  - If it is an expression, then we introduce a self-loop transition in each location of the device automaton with the expression ϕ as the guard. The labels of the added device transition and in the rule model are both $Sensor\_Rule_i$.
- It is normal to see a time delay between the enabling of the triggering condition and the execution of the corresponding API. Such delay can be marked as the invariants and guards in the model. For example, in Figure 3(c), we have invariant $t \leq 5$ on location *waiting* and guard $t \geq 3$ on the transition $Close\_Rule_1$. This means the potential delay is $3 \leq t \leq 5$.

Let's recall the IFTTT rule for the CO example given in Section 1:

```
IF Smart_Fan.CO_reading == 195 THEN execute Alarm.TURN_ON command
IF Alarm.TURN_ON.Signal ==TRUE THEN execute Smart_Fan.ACTIVATE command
```

After processing the JSON schema, Figure 2, and the rules of the CO example, the corresponding LHA models generated according to these rules are shown in Figure 3 (a-d), respectively.

4.3 Feasibility of Verifiable Schema

The modeling process can apply to a wide range of IoT devices, and both the industry and academia have been pushing hard in this direction. For example, the AllSeen alliance (which is backed up 50+ member companies such as Microsoft, Qualcomm, NetGear, HoneyWell, and LG) has a working group called ÒCommon Device ModelsÓ. Similarly, GoogleÕs Weave protocol also mandates that compatible IoT devices provide device schemas. Furthermore, from the aspect of infrastructure, AmazonÕs AWS offers IoT device registry services.

Building on this momentum in the industry, our contribution is to highlight those device specifications that would be necessary for policy verification. In fact, most of these specifications are not difficult for IoT device manufacturers to provide. Furthermore, we discuss how these device specifications should be used to achieve our goals.

5 SYSTEM CHECKING AND FIX SUGGESTION

The previous section presents a method to construct LHA models for a given set of HA-IoT devices and IFTTT-style automation rules. Now, we discuss approaches to efficiently check whether these models conform to specifications and systematically synthesize solutions to resolve identified violations.

5.1 Specification Authoring and Specification-Related Reachability Checking

In addition to the system model, the HA-IoT programming tool suite must allow IoT users to author confidence/reachability specifications. We note that these specification should be easy for nonexpert users to read and write. To this end, we define the specification format as a conjunction of conditions that *shall not happen*. This specification format is intuitive to nonexpert IoT users as it is similar to IFTTT-style programming. In our CO example, if users do not want the room CO concentration to be higher than 200, they can write `SmartFan.CO ≥ 200`.

After we get specifications from users, we can check that the system of LHA models can never reach a given undesirable state. One option is to conduct BMC checking by directly using off-the-shelf checkers. However, this option can have a significant overhead because these checkers always explore the entire state space of devices and rules, regardless of whether there are meaningful interactions among devices. At the same time, we note that the performance of state reachability checking can degrade quickly as the number of components increases. Therefore, we propose to only consider the subset of models that are related through specifications, to shrink the state space for the underlying checker. This approach is formally presented next.

Definition 5.1. Given a composed LHA network $N = H_1||H_2|| \ldots ||H_m$, and a reachability specification $\mathcal{R}(v, \varphi)$, if v is a location of H_i, φ consists of variables from H_j, $(1 \leq i, j \leq m)$, we say H_i and H_j are $\mathcal{R}$ *related*.

Definition 5.2. If two LHA models have a shared label, we say these two models are *related*. Given three LHA models, A, B, and C, if A is related to B, B is related to C, then A is related to C. Given a composed LHA network $N = H_1||H_2|| \ldots ||H_m$, we call the sets of all the LHA models related to H_i the *related closure* of H_i $(1 \leq i \leq m)$.

Definition 5.3. Given a composed LHA network $N = H_1||H_2|| \ldots ||H_m$, and a reachability specification $\mathcal{R}(v, \varphi)$, if v is a location of H_i, φ consists of variables from H_j, $(1 \leq i, j \leq m)$, we say H_i and H_j are $\mathcal{R}$ *related*. Then, the set of LHA models consists of the related closure of H_i and the related closure of H_j is the $\mathcal{R}$ *related closure*.

Technically, given a reachability specification $\mathcal{R}(v, \varphi)$, we first compute the $\mathcal{R}$ *related closure* subset of models. Then, we feed it to the underlying checker to do the checking. In this manner, since the size of the system under checking is reduced, checking can be done with a smaller overhead.

5.2 Counterexample-Guided Fix Suggestion

The output of BMC checking indicates whether the set of models passes or fails the verification. If models fail the check, it means there exists a sequence of transitions that can reach an undesirable state. In another word, bad things could happen. For IoT users, this potentially undesirable scenario should be resolved before the automation rule set is deployed.

Again, it is impractical to count on an average end-user to fix the system based on his or her understanding of the counterexample trace. Even for a domain expert, debugging and fixing are not trivial. Because our framework is user-facing, there will not be an expert to help the user to analyze and fix violations. Therefore, such tasks should be automated.

To resolve a specification violation, the first thing is to identify the problematic automation rule(s). Different from debugging for general CPS software, IoT users can realistically change only the automation rules, rather than changing the installed IoT devices or specifications. Specifically, we can only adjust the triggering condition value of IFTTT rules.

To systematically perform this task, we propose parameterizing IFTTT rules and then solving for solutions to the parameterized system. Each of these solutions would represent one valid configuration that can be presented to the IoT user.

In our CO example, suppose the reachability specification asks whether the CO concentration in the room can ever exceed 200. The checker finds a counterexample: As the smart fan detects the CO level reaching 195, rule 1 is executed. However, as there is a delay between the satisfaction of

ALGORITHM 1: Counterexample-Guided Fix Suggestion

```
1:  procedure CE–Analysis (Counterexample Path $\rho$, Specification $\mathcal{R}(v, \varphi)$, Rule Set $RS$)
2:      Encoding the reachability of $\rho$ according to $\mathcal{R}(v, \varphi)$ as Formula $\Psi$
3:      Denote the constraints related with all the rules in $RS$ as $\Theta$ and the other rules in $\Psi$ as $\Phi$
4:      Therefore, $\Psi = (\bigwedge_{i=1}^{n} \phi_i) \wedge (\bigwedge_{i=1}^{m} \theta_i)$, where $\phi_i \in \Phi$, and $\theta_i \in \Theta$
5:      for each $\theta_i$, $(Device_i.variable_j == concrete\_value_k)$, $\in \Theta$ do
6:          Parameterize $concrete\_value_k$ to a free parameter $para_k$
7:          Generate new constraint $\theta' = (Device_i.variable_j == para_k)$
8:      end for
9:      Generate formula $\Theta' = \bigwedge_{i=1}^{m} \theta'_i$, and $\Psi' = (\bigwedge_{i=1}^{n} \phi_i) \wedge \Theta'$
10:     Take the negation of all the subformulas $\psi_i$ in $\Psi'$, get new formula $\Psi'' = \bigvee \neg\psi_i$
11:     Denote all the variables in $\Psi$ as $Vari$
12:     Use QE to check: Whether $\exists para_k$, such that $\forall x_i \in Vari$, $\Psi''$ is feasible.
13:     QE returns the value range for each $para_k$, which is the suggestion of the fix
14: end procedure
```

the triggering condition and the execution of the triggered command, the CO concentration can reach 200 before the smart fan is activated. The original rule says `IF Smart_Fan.CO == 195 THEN Execute Alarm.TURN_ON`. We parameterize the rule to `IF Smart_Fan.CO == A THEN Execute Alarm.TURN_ON`. Then, we need to solve for values of A that can invalidate the specification.

Instead of computing the potential range of A directly, we propose a *counterexample-guided approach*. Our basic idea is to find values of A to dismiss the found counterexample path ρ. The algorithm is shown in Algorithm 1. As introduced in Section 2, given a path in the model, whether that path satisfies the specification is encoded to the feasibility of a formula Ψ. Then, we parameterize the threshold of the triggering conditions in the rules to free parameters $para_k$, and we modify Ψ to Ψ' accordingly (Lines 5–9).

We can reformulate the problem thus: Can we find a valuation for these parameters to make Ψ' infeasible? If the answer is yes, then the located counterexample is dismissed. We take the negation of all the subformulas in Ψ', make a disjunction of them, and get new formula Ψ''. Now, the question become whether we can find $para_k$ to make Ψ'' feasible (Line 10).

As all constraints in Ψ are linear, we can use QE (Monniaux 2008) to transform this problem to an *equivalent* quantifier-free numerical formula about $para_k$. This formula gives the value range for each $para_k$ that can make Ψ' infeasible (in other words, dismiss the found counterexample; Line 11–13). Clearly, we can simply select a value in the range as a fix suggestion and ask the checker to check the system again. As the number of potential paths in the given bound is finite, this procedure is guaranteed to terminate.

Continuing the CO example, we first parameterize the condition `Smart_Fan.CO == 195` to `Smart_Fan.CO == A`. Then, we conduct the negation and QE procedures, as presented. The resulting formula for A from the QE solver is $0 \leq A \leq 190$, and the Satisfiability Modulo Theories (SMT) solver selects a value from this range (e.g., 165). We use it as a potential fix and check the set of models again. The system should pass the verification this time, and it subsequently presents the new value as a fix suggestion to the user.

5.3 Handling of Conditions of Inequalities

In the last subsection, the conditions of the triggers are all presented as equalities. The problem with inequality constraints is a rather complex but still can be handled in a similar way.

According to the "lazy" semantic of LHA, when a transition guard is an inequality, the transition can be fired at any time spot that satisfies the inequality guard. In other words, the time spot where the transition is fired is not required to be the exact time spot that satisfies the guard. Take the CO level, for example: The transition guard `Smart_Fan.CO ≥ 195` can be fired when CO level ≥ 1,000 because this still satisfies the semantics of the model.

As a result, even if we modify the condition to `Smart_Fan.CO ≥ 165`, as we did in the equality case, the model checker can still find a behavior which fires the trigger too late, when the CO level is too high, say, 1,000. In other words, if the triggering condition is a half interval, the limitation of LHA semantics makes it difficult to control the timing at which the transition will be fired.

Therefore, when our method needs to fix conditions with inequalities of half-intervals, it changes it to a parametric interval first. Still using the CO example, if the original condition is `Smart_Fan.CO ≥ 165`, the parameterized condition is `b≥ Smart_Fan.CO ≥ a`, where a and b are free parameters. Then, Line 12 of Algorithm 1 is changed to "Use QE to check: Whether $\exists a$, $\exists b$, such that $\forall x_i \in Vari$, Ψ'' is feasible." Then the following procedure is the same with equality.

In the CO example, the QE result we get is $300 \geq a \geq 0$, $190 \geq b \geq 0$, and $b \geq a$. The SMT solver selects values for a and b from the corresponding range (e.g., a = 75, b = 180). We modify the condition to `180 ≥ Smart_Fan.CO ≥ 75`. We use it as a potential fix, and the system then passes the verification.

6 SYSTEM IMPLEMENTATION AND EVALUATION

This section is organized by the following major results. First, the technique proposed and discussed in this article is implemented in a tool, MenShen. Second, our optimization techniques allow MenShen to gracefully scale with LHA size. Empirical results suggest that, for a deployment of up to 46 devices and 65 rules, MenShen can finish within 10 seconds (i.e., the typical human attention span (Nielsen Norman Group 1993)). Third, a user study confirms that most users cannot find the bug and fix the system. Meanwhile, the user study also suggests that more than 66.7% of fix suggestions are accepted without further user intervention. The implementation of MenShen and all experimental data are available online (Bu 2016).

6.1 System Implementation Overview

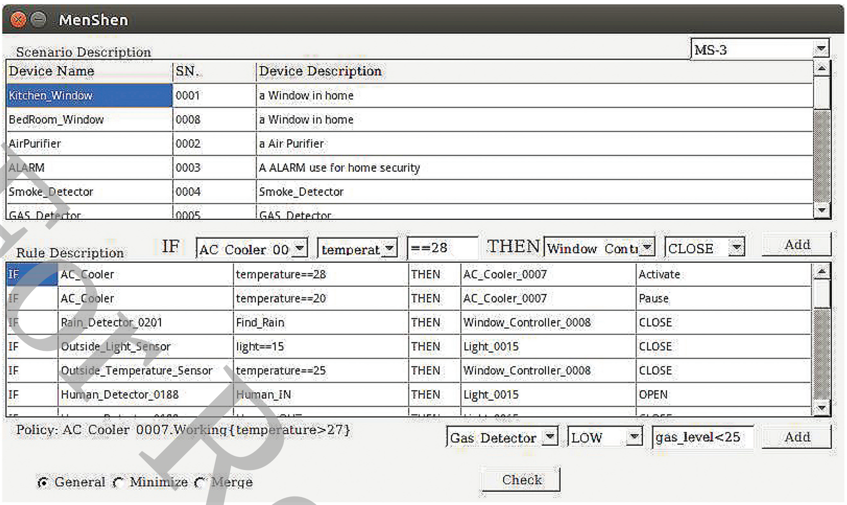
Our current system implementation, MenShen,[5] supports functionalities discussed in previous sections: (1) automated LHA model generation from device schemas, (2) template-based GUI for authoring IFTTT-style rules and specifications, (3) automated system reachability checking, and (4) violation fix suggestions.

MenShen is implemented in C#. Some system components are based on third-party libraries: BACH (Bu 2006; Bu et al. 2008) for the LHA checker, Redlog (Dolzmann 2006) for the QE solver, Z3 (de Moura and Bjørner 2008) for the SMT solver, and Json.Net (jso 2009) for parsing the device schemas.

Here we show the GUIs of the system to demo the functionalities of MenShen. The main GUI is shown in Figure 4, where we can see that a user can specify rules and the specifications using drop-down list-style templates. MenShen reads the device information directly from their docs. Then, users can select the name of the device, the observable variables, and the APIs allowed to call to compose a rule/specification directly.

We can also find three options for the user to select before they click the check button at the bottom of Figure 4. The options include:

[5] MenShen is the name of the door god in Chinese.



MenShen

Scenario Description   MS-3

Device Name	SN.	Device Description
Kitchen_Window	0001	a Window in home
BedRoom_Window	0008	a Window in home
AirPurifier	0002	a Air Purifier
ALARM	0003	A ALARM use for home security
Smoke_Detector	0004	Smoke_Detector
GAS_Detector	0005	GAS_Detector

Rule Description   IF AC Cooler 00   temperat   ==28   THEN Window Conti   CLOSE   Add

IF	AC_Cooler	temperature==28	THEN	AC_Cooler_0007	Activate
IF	AC_Cooler	temperature==20	THEN	AC_Cooler_0007	Pause
IF	Rain_Detector_0201	Find_Rain	THEN	Window_Controller_0008	CLOSE
IF	Outside_Light_Sensor	light==15	THEN	Light_0015	CLOSE
IF	Outside_Temperature_Sensor	temperature==25	THEN	Window_Controller_0008	CLOSE
IF	Human_Detector_0188	Human_IN	THEN	Light_0015	OPEN

Policy: AC Cooler 0007.Working{temperature>27}   Gas Detector   LOW   gas_level<25   Add

● General ○ Minimize ○ Merge   Check

Fig. 4. Main GUI of MenShen.

- General: No optimization technique is applied.
- Minimize: Perform checking on the property-related closure, as described in Section 5.1.
- Merge: Link and analyze the rules with the same triggering condition together. This prevents MenShen from assigning different thresholds while generating fix suggestions.

If MenShen finds a violation of the specification, the fixing procedure is activated. We grant users the option to mark rules that should never be changed. They can also provide the preferred range for individual variables. Then MenShen will look for a solution in the user-specified range. When multiple fix suggestions are possible, MenShen prioritizes the suggestions based on how similar they are to the original rules. Minimizing this difference can improve user acceptance of fix suggestions. The related GUI is shown in Figure 5.

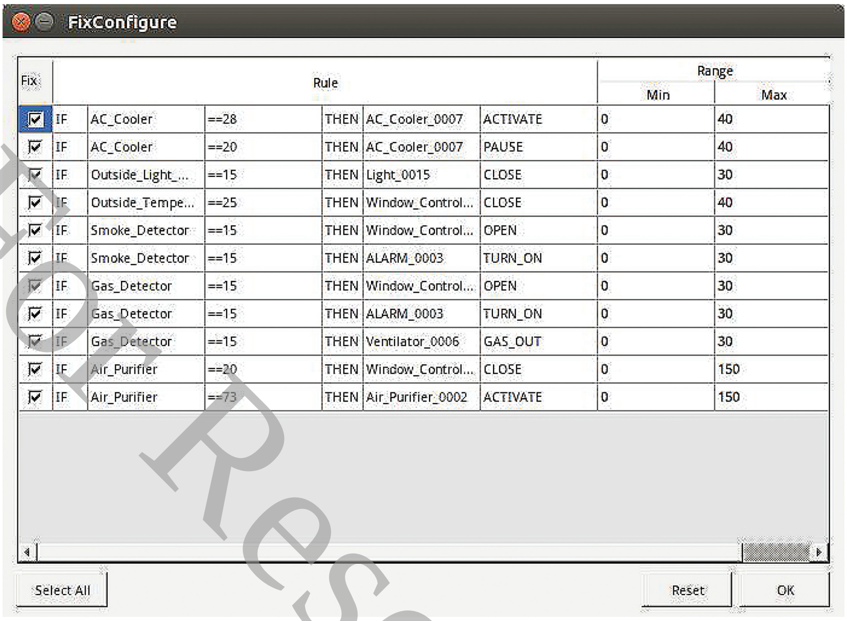
After the fix suggestion is synthesized, MenShen will conduct a new round of confirmation checking. If the new system passed the confirmation check, MenShen pushes the fix suggestion to the user, as we can see in Figure 6. The changed rules are marked in red. Last but not least, if users are not satisfied with the fix suggestion, they are free to modify the rule and start the procedure again.

6.2 Real-World Evaluation Datasets

Our nine datasets contain automation rule sets deployed in the real world. They are from three sources: (1) four small-scale systems (labeled SC-1, SC-2, SC-3, SC-4) are based on rules shared by normal IFTTT.com users (ift 2011); (2) four large real-world HA-IoT systems are from office deployments (labeled MS-1, MS-2, MS-3, MS-4); and (3) one is used by a related effort (Universal Devices Products 2007) (Croft et al. 2015) (labeled ISY).

SC-1, SC-2, SC-3, and SC-4 have 2-3 devices and rules. Each of MS-1, MS-2, MS-3, and MS-4 has tens of rules and devices, similar to typical home automation systems. The ISY dataset has 46 devices and 65 rules. In total, there are 26 different types of devices including gas meters, HVAC, lights, air purifiers, GPS, and water heaters.

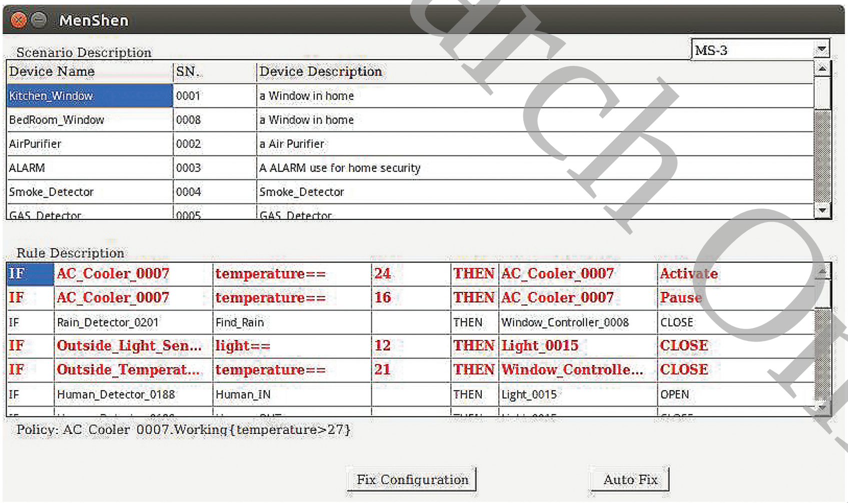
Table 1 lists the most important system property that dataset owners expect their automation rule sets to comply with.



FixConfigure

Fix	Rule						Range	
							Min	Max
☑	IF	AC_Cooler	==28	THEN	AC_Cooler_0007	ACTIVATE	0	40
☑	IF	AC_Cooler	==20	THEN	AC_Cooler_0007	PAUSE	0	40
☑	IF	Outside_Light_...	==15	THEN	Light_0015	CLOSE	0	30
☑	IF	Outside_Tempe...	==25	THEN	Window_Control...	CLOSE	0	40
☑	IF	Smoke_Detector	==15	THEN	Window_Control...	OPEN	0	30
☑	IF	Smoke_Detector	==15	THEN	ALARM_0003	TURN_ON	0	30
☑	IF	Gas_Detector	==15	THEN	Window_Control...	OPEN	0	30
☑	IF	Gas_Detector	==15	THEN	ALARM_0003	TURN_ON	0	30
☑	IF	Gas_Detector	==15	THEN	Ventilator_0006	GAS_OUT	0	30
☑	IF	Air_Purifier	==20	THEN	Window_Control...	CLOSE	0	150
☑	IF	Air_Purifier	==73	THEN	Air_Purifier_0002	ACTIVATE	0	150

Fig. 5. GUI for fix configuration.



MenShen

Scenario Description

MS-3

Device Name	SN.	Device Description
Kitchen_Window	0001	a Window in home
BedRoom_Window	0008	a Window in home
AirPurifier	0002	a Air Purifier
ALARM	0003	A ALARM use for home security
Smoke_Detector	0004	Smoke_Detector
GAS_Detector	0005	GAS_Detector

Rule Description

IF	AC_Cooler_0007	temperature==	24	THEN	AC_Cooler_0007	Activate
IF	AC_Cooler_0007	temperature==	16	THEN	AC_Cooler_0007	Pause
IF	Rain_Detector_0201	Find_Rain		THEN	Window_Controller_0008	CLOSE
IF	Outside_Light_Sen...	light==	12	THEN	Light_0015	CLOSE
IF	Outside_Temperat...	temperature==	21	THEN	Window_Controlle...	CLOSE
IF	Human_Detector_0188	Human_IN		THEN	Light_0015	OPEN

Policy: AC_Cooler_0007.Working{temperature>27}

Fig. 6. GUI for fix suggestion.

6.3 System Scalability

MenShen aims to minimize the user burden in realizing high-confidence real-time HA-IoT systems. Given that formal checking techniques are mature enough to accurately identify policy violations, this section discusses system scalability in IoT scenarios. Specifically, we show that optimization techniques allow MenShen to exhibit a processing latency of less than the typical 10-sec user attention span (Nielsen Norman Group 1993).

Table 1. System Properties that Dataset Owners Expect Their Automation Rule Sets to Satisfy

System	Policy
SC-1	The hot water is ready when the user is back home.
SC-2	The temperature of water in the bathtub should not drop down to 40 degree when the user is back home.
SC-3	The level of CO in the room should never be dangerous.
SC-4	The home security should be closed when the garage door is opened.
MS-1	The level of smoke in the room should never be too high.
MS-2	The level of gas should never be dangerous.
MS-3	The temperature in the room should stay below 27.
MS-4	The level of PM2.5 in the room should not be harmful.
ISY	The light in the bedroom should be turned off at 8:00 pm.

Table 2. Results of Checking and Fixing Real-Case HA IoT Systems by MenShen

			Original System				System Minimization				Related Rules Merging			
System	#Devices	#Rules	#LHA	Model(s)	Check (s)	Fix (s)	#LHA	Model(s)	Check (s)	Fix (s)	#LHA	Model(s)	Check (s)	Fix (s)
SC-1	3	3	6	0.011	0.26	0.98	6	0.011	0.26	0.98	6	0.012	0.26	0.98
SC-2	3	3	6	0.011	0.26	1.25	6	0.010	0.26	1.25	6	0.011	0.26	1.25
SC-3	2	2	4	0.011	0.12	0.66	4	0.011	0.12	0.66	4	0.013	0.12	0.66
SC-4	3	2	5	0.010	0.22	0.92	5	0.013	0.22	0.92	5	0.013	0.22	0.92
MS-1	12	15	27	0.013	1.89	51.17	20	0.012	0.88	3.62	20	0.012	0.92	3.32
MS-2	12	15	27	0.012	1.79	365.58	20	0.013	1.23	8.62	20	0.015	1.02	7.88
MS-3	18	20	38	0.012	3.77	11.19	3	0.017	0.23	0.97	3	0.013	0.23	0.95
MS-4	18	19	37	0.013	3.56	10.5	3	0.012	0.23	0.98	3	0.013	0.23	0.95
ISY	46	65	111	0.014	9.25	1209.4	25	0.024	0.97	7.51	25	0.016	0.52	3.98

#Devices and #Rules denote the number of devices and rules in the dataset, respectively. #LHA denotes the number of generated LHA models. Check denotes the time spent in checking the problem. Fix denotes the time spent in fixing. The default bound set for all problems is 10 for all the Automata.

Experiments were conducted by checking whether individual datasets satisfy their expected system property listed in Table 1. We used a ThinkCenter workstation, with Intel Core 2 Quad CPU Q9500 @ 2.83GHz × 4, 4GB RAM, and Ubuntu 14.04 64-bit.

MenShen Implementation. Table 2 shows the empirical results under three different MenShen settings: "Original System" refers to running MenShen without additional constraints, "System Minimization" refers to running the specification verification with only related models, "Related Rules Merging" refers to manually marking rule variables that should be the same (as discussed in Section 6.1). As we discuss next, the system size (i.e., the number of LHA states) largely determines the system latency, and we also evaluated the effectiveness of two optimization techniques.

If the size of the system increases significantly, QE can be the performance bottleneck. Specifically, for the four small cases (SC-1, SC-2, SC-3, and SC-4), MenShen can complete both the checking and fix suggestions in less than 1 second. However, for the ISY dataset, which is 20 times larger than the SC series cases, the latency increases quickly to 1,209 seconds.

As discussed next, optimization techniques can reduce system latency for large cases. Table 2 shows two such optimizations: System Minimization and Related Rules Merging.

First, when the system size is large, especially if not all the devices are connected with each other, the System Minimization technique exhibits a significant improvement space. Specifically, the number of automata in MS-3 is reduced from 38 to 3, and the number of automata in the ISY case is reduced from 111 to 25. Since the entire state space is reduced significantly, the performance

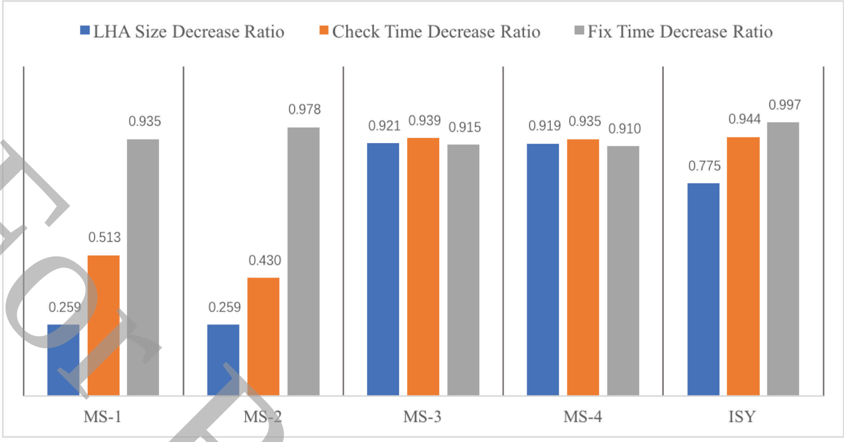


Fig. 7. Optimization evaluation: Related rules merging VS, no optimization.

of MenShen improves substantially in all five large datasets. And the time for ISY is reduced from more than 1,209 seconds down to only 7.5 seconds. However, we note that since SC-1, SC-2, SC-3, and SC-4 are small datasets, the systems are very compact. Therefore, the gain from optimization techniques is negligible.

Second, we look at the gain from the Related Rules Merging technique. When the number of rules is large, it is not rare to have different rules sharing the same trigger conditions. In this case, while the number of automata under check stays the same, the structure for the related models can be simplified to reduce the number of parameters to solve. As this optimization contains system minimization, we report the decrease ratio of "Related Rules Merging" versus no optimization on the five large problems concerning LHA size, time for checking, and time for fix, respectively, in Figure 7. We can see that the optimization methods work very well in that up to 99.7% of time can be saved on large systems like ISY.

6.4 User Study

To understand the usefulness of MenShen's feedback to IoT users, we conducted a set of user studies with our real-world automation rule sets. Specifically, we took the MS-4 dataset, which represents a room with 20 IoT-enabled devices and 18 automation rules. And we added two more system properties to check to the one in Table 2.

We have two group of volunteers. Each group has 45 participants. The study participants in the first group include researchers and interns from Microsoft Research Asia, and PhD and master's degree students in the software engineering discipline at Nanjing University. The participants in the second group include non-computer science (CS) major college students, high school students, and also some housewives.

We asked the participants to decide whether the connected IoT system can violate any of the three given specifications, and then we asked participants to attempt to fix these violations manually. Table 3 summarizes the user study results of the CS-Major users, while the results of Non-CS-Major users are presented in Table 4.

For comparison, these tables also include the performance of MenShen (without any optimization enabled). Several observations support the effectiveness of MenShen. For example, Tables 3 and 4 suggest that the majority of participants cannot find any violation after spending 1 to 8 minutes searching, and only four participants (two from the CS-Major Group and, two from

Table 3. User Study on MS-4 Scenario from CS-Majored Students and Researchers

Problem	Total Partici.	Average Total Time(s)	Average Check(s)	Average Fix(s)	Able to Find Conflicts	Able to Fix	MenShen Check(s)	MenShen Fix(s)	User Acceptance
Q1	45	453	444	82	7/45	2/45	3.56	10.5	39/45
Q2	45	98	90	63	7/45	2/45	3.82	17.77	37/45
Q3	45	75	63	57	11/45	2/45	3.14	9.72	45/45

Average time denotes the average time spent by the participants in analyzing the specific problem. MenShen time denotes the time spent by MenShen in the same problem.

Table 4. User Study on MS-4 Scenario from Non-CS-Majored Participants

Problem	Total Partici.	Average Total Time(s)	Average Check(s)	Average Fix(s)	Able to Find Conflicts	Able to Fix	MenShen Check(s)	MenShen Fix(s)	User Acceptance
Q1	45	326	443	61	14/45	5/45	3.56	10.5	30/45
Q2	45	85	70	26	6/45	2/45	3.82	17.77	33/45
Q3	45	48	47	32	13/45	10/45	3.14	9.72	31/45

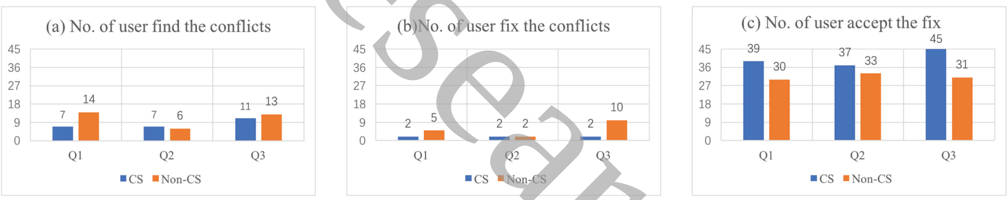


Fig. 8. Data of CS major participants versus Non-CS major participants.

the Non-CS-Major group) were able to identify and fix all the problems successfully. We can see that, for the majority of participants, even for users with a background in computer science, successfully realizing a high-confidence HA-IoT system is still a difficult task.

We've also organize and present the user study data of CS-Major users and Non-CS-Major users in Figure 8. Interestingly, the percentage of users who can find and fix the system is higher for Non-CS-Major than for CS-Major participants in many cases, as shown in Figure 8(a,b). This supports our observation that, while the popular IFTTT-style programming paradigm simplifies authoring, it does not simplify the task of checking and fixing specification violations. For average users of HA-IoT, even CS background education cannot alleviate this problem.

Delving into Tables 3 and 4, we discuss the user acceptance rate to MenShen feedback. Specifically, this metric is quantified by presenting MenShen's fix suggestions to the participants. Tables 3 and 4 show that, after we explained to the user the reason the system failed and fixed the related rule, user acceptance of the fix suggestions made by MenShen is in the range of 66.7%(30/45) to 100% (45/45). One of the main reasons that some participants are not satisfied with the fix suggestion is that they have difficulty understanding how the original parameter may have caused the error and why the changed version is correct. Therefore, we can see from Figure 8(c) that the user acceptance rate from the CS-Major group is higher than from the Non-CS-Major group in general. This raises an interesting question about how to present failures to users in the future.

In addition to these findings, one lesson learned is that, while several scenarios in a space can be programmed by a set of automation rules, IoT users tend to focus on manually verifying one scenario at a time. While this approach reduces the manual burden, it ignores interactions among scenarios. For example, turning on the HVAC can cause the air ventilation system to stop intaking outside air in the summer, which can have a undesirable consequence depending on indoor CO concentration. This observation brings up the value of having programming assistance from MenShen.

7 DISCUSSION

In this work, we demonstrate the feasibility of using LHA for HA-IoT systems. Not only does LHA simplify the presentation of device communications and the dynamic behavior of analog environments, it also efficiently checks their complex behavior. We now describe system limitations that are beyond the scope of this article.

First, we assume that device models are static and do not change over a short period of time. If the modeled environment is highly dynamic, parametric models (Bu et al. 2011) may yield better results. Furthermore, we note that it can be costly to model continuous analog environments that are under the influence of multiple IoT devices. For example, room temperature can be affected by outdoor temperature and indoor appliances. Future work will focus on reducing this cost on the underlying system infrastructure. Second, the semantics of transition in LHA is not "Urgent" (Schupp et al. 2015). Therefore, we may encounter situations where the trigger condition is enabled but the model does not fire such a transition. Third, we currently do not consider human-in-the-loop or user models that can change analog environments at any time. We leave this for future work.

8 RELATED WORK

IoT Software System Checking and Monitoring. The high-confidence analysis of IoT systems has recently gained attention in the community. SIFT (Liang et al. 2015) took the first step of demonstrating the potential of correctness checking of IoT systems, and it used the symbolic execution method to generate test cases to test the abstracted code of an IoT system. In contrast to MenShen, SIFT assumes IoT users have the necessary knowledge and background to run such procedures. Furthermore, SIFT does not consider the temporal behavior of devices nor violation debugging.

Like MenShen, DeLorean (Croft et al. 2015) argues the importance of modeling the temporal behavior when checking HA-IoT systems. They proposed building a timed automata model for home automation control programs. However, they assumed a manual modeling procedure, which is not practical for nonexpert IoT users. Furthermore, timed automata can only model a time clock with uniform speed, rather than any continuous variables with arbitrary clocks. Therefore, MenShen can theoretically handle a wider spectrum of HA-IoT scenarios than DeLorean. Last but not least, they stop DeLorean after the checking is finished, but our work continues to synthesize fix suggestions.

DepSys (Munir and Stankovic 2014) presented a method to specify and check the dependency of devices in a home automation IoT system. However, they only focus on potential conflicts among the devices, say, A and B control the same device. This solution does not address system-wide, and especially time-related, policy violations like MenShen.

In addition to these efforts in correctness checking, there are also investigations in the area of invariant correctness monitoring (Gună et al. 2014; Herbert et al. 2007). Generally, these works perform online monitoring of invariants concerning certain parameterÕs values to see whether certain values will break the invariant during system operation. Then, if an invariant violation is detected, predefined safety-related rules could be called in a way similar to IFTTT.

These works mainly focus on the efficiency of runtime detecting certain invariant violations (say, catch the threshold in a timely manner). By contast, our work tries to make sure that such violations will not happen. We perform an offline formal verification style check to guarantee correctness, and we also help to fix the rules when the original system cannot meet the specification.

Parameter Fix Suggestion. The fix suggestion synthesis performed by MenShen is related to the classical parameter synthesis problem. This problem has been amply studied already. Studies by Henzinger and Wong-Toi (1995) and Frehse et al. (2008) are the closest works to MenShen as they all deal with real-time hybrid automata.

The problem of parameter synthesis for LHA was already proposed (Henzinger and Wong-Toi 1995). However, in that solution, the whole reachable set had to be computed first, which is very expensive. Thus, the systems they can handle are rather limited.

Similar with this work, Frehse et al. (2008) also works on parameter synthesis for LHA. They propose a CEGAR framework to find the values for the parameters which can avoid "bad" states in the complete state space. As MenShen is performing BMC rather than general MC, we are facing a much smaller state space. Therefore, we can use QE directly on the counterexample path to find potential parameter assignments.

Recently, a study (Cimatti et al. 2013) proposed a method to perform optimal parameter synthesis for infinite state space systems. These researchers extended the IC3 (Bradley 2011) framework to compute the precise region incrementally. This is an interesting work, and we will try to adapt it into MenShen in future work.

9 CONCLUSION

This article presents MenShen, a novel framework of automated end-to-end programming assistance, to help nonexpert IoT users in systematically realizing high-confidence real-time HA-IoT systems. In contrast to related efforts that handle only high-level logic, MenShen can model and reason about both real-time behavior and analog environments. Furthermore, not only does MenShen check whether an automation rule set violates specifications, it also effectively suggests possible solutions to users.

As future work, we will investigate methods to model continuous aspects that are influenced by multiple devices and environmental factors. Personalized parameter generation is also an interesting topic which can help to increase user satisfaction. Furthermore, leveraging probabilistic model checking to generate user-friendly quantitative probabilistic reports of such HA-IoT systems is also worth investigation.

APPENDIX

In this section, we review the path-oriented reachability checking encoding presented in earlier work (Li et al. 2007; Bu and Li 2011). This technique is the underlying decision procedure of MenShen for reachability checking.

For a path in an LHA H of the form $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$, by assigning each location v_i with a time delay stamp δ_i we get a *timed sequence* of the form $\langle {v_0 \atop \delta_0} \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle {v_1 \atop \delta_1} \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle {v_n \atop \delta_n} \rangle$ where δ_i $(0 < i \leq n)$ is a non-negative real number and $\delta_0 = 0$ as $v_0 = v_I$ is the initial location. This time sequence represents a behavior of H such that the system starts from the initial location v_0; stays there for δ_0 time units, which is 0; then jumps to v_1 and stays at v_1 for δ_1 time units; and so on.

The behavior of an LHA can be described informally as follows. The automaton jumps from the initial location v_0 to v_1 to initialize all the variables. Then, as time progresses, the values of all variables change continuously according to the flow condition associated with the current location. At any time, the system can change its current location from v to v' provided that there is a transition $(v, \sigma, \phi, \psi, v')$ from v to v' whose transition guards in ϕ are all satisfied by the current value of the variables. With a location change by a transition $(v, \sigma, \phi, \psi, v')$, some variables are reset to the new value according to the reset actions in ψ. Transitions are assumed to be instantaneous.

Let $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$ be an LHA. Given a timed sequence ω of the form $\langle \substack{v_0\\ \delta_0} \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle \substack{v_1\\ \delta_1} \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \dots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle \substack{v_n\\ \delta_n} \rangle$, let $\zeta_i(x)$ represent the value of x ($x \in X$) when the automaton has stayed at v_i for delay δ_i and $\lambda_i(x)$ represent the value of x at the time the automaton reaches v_i along with ω ($0 \le i \le n$). It follows that

$$\lambda_{i+1}(x) = \begin{cases} d & \text{if } x := d \in \psi_i \\ \zeta_i(x) & \text{otherwise} \end{cases} \quad (0 \le i < n).$$

Definition 9.1. For an LHA $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$, a timed sequence of the form $\langle \substack{v_0\\ \delta_0} \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle \substack{v_1\\ \delta_1} \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \dots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle \substack{v_n\\ \delta_n} \rangle$ represents a behavior of H if and only if the following condition is satisfied:

- $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \dots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ is a path;
- each variable $x \in X$ evolves according to its flow condition in each location v_i ($0 < i \le n$); that is, $u_i\delta_i \le \zeta_i(x) - \lambda_i(x) \le u'_i\delta_i$ where $\dot{x} \in [u_i, u'_i] \in \beta(v_i)$;
- all the transition guards in ϕ_i ($1 \le i \le n - 1$) are satisfied; that is, for each transition guard $a \le \sum_{k=0}^{l} c_k x_k \le b$ in ϕ_i, $a \le \sum_{k=0}^{l} c_k \zeta_i(x_k) \le b$;
- the location invariant of each location v_i ($1 \le i \le n$) is satisfied; that is, at the time the automaton reaches and leaves v_i, each constraint $a \le \sum_{k=0}^{l} c_k x_k \le b$ in $\alpha(v_i)$ ($1 \le i \le n$) is satisfied (i.e., $a \le \sum_{k=0}^{l} c_k \lambda_i(x_k) \le b$ and $a \le \sum_{k=0}^{l} c_k \zeta_i(x_k) \le b$).

Definition 9.2. For an LHA $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$, if a timed sequence of the form $\langle \substack{v_0\\ \delta_0} \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle \substack{v_1\\ \delta_1} \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \dots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle \substack{v_n\\ \delta_n} \rangle$ is a behavior of H, we say path $\rho = \langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \dots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ is *feasible*, and location v_n is *reachable* along ρ.

Definition 9.3. Let $H = (X, \Sigma, V, v_I, E, \alpha, \beta)$ be an LHA, and $\mathcal{R}(v, \varphi)$ be a reachability specification. A behavior of H of the form $\langle \substack{v_0\\ \delta_0} \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle \substack{v_1\\ \delta_1} \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \dots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle \substack{v_n\\ \delta_n} \rangle$ *satisfies* $\mathcal{R}(v, \varphi)$ if and only if $v_n = v$ and each constraint in φ is satisfied when the automaton has stayed in v_n for delay δ_n; that is, for each variable constraint $a \le \sum_{k=0}^{l} c_k x_k \le b$ in φ, $a \le \sum_{k=0}^{l} c_k \zeta_n(x_k) \le b$ where $\zeta_n(x_k)$ ($0 \le k \le l$) represents the value of x_k when the automaton has stayed at v_n for the delay δ_n. H *satisfies* $\mathcal{R}(v, \varphi)$ if and only if there is a behavior of H which satisfies $\mathcal{R}(v, \varphi)$.

According to Definitions 9.2 and 9.3, the reachability of a given path in an LHA model can be encoded to the feasibility of a conjunction of a set of linear constraints, which can be solved efficiently by Linear Programming SMT techniques.

REFERENCES

Apple HomeKit. 2016. Apple HomeKit. Retrieved from http://www.apple.com/ios/homekit/.

Gilles Audemard, Marco Bozzano, Alessandro Cimatti, and Roberto Sebastiani. 2005. Verifying industrial hybrid systems with mathSAT. *Electrical Notes on Theoretical Computer Science* 119, 2 (2005), 17–32.

AWS IoT. 2015. Device Registry for AWS IoT. Retrieved from http://docs.aws.amazon.com/iot/latest/developerguide/thing-registry.html.

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. 2003. Bounded model checking. *Advances in Computers* 58 (2003), 117–148.

Aaron R. Bradley. 2011. SAT-based model checking without unrolling. In *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'11).* 70–87.

Lei Bu. 2006. BACH. Retrieved from http://seg.nju.edu.cn/BACH/.

Lei Bu. 2016. MenShen Project Page. Retrieved from http://seg.nju.edu.cn/MenShen/.

Lei Bu and Xuandong Li. 2011. Path-oriented bounded reachability analysis of composed linear hybrid systems. *International Journal on Software Tools for Technology Transfer* (2011), 307–317.

Lei Bu, You Li, Linzhang Wang, and Xuandong Li. 2008. BACH: Bounded reachability checker for linear hybrid automata. In *Proceedings of Formal Methods in Computer-Aided Design (FMCAD'08).* 1–4.

Lei Bu, Qixin Wang, Xin Chen, Linzhang Wang, Tian Zhang, Jianhua Zhao, and Xuandong Li. 2011. Toward online hybrid systems model checking of cyber-physical systems time-bounded short-run behavior. *SIGBED Review* 8, 2 (2011), 7–10.

Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. 2013. Parameter synthesis with IC3. In *Proceedings of Formal Methods in Computer-Aided Design (FMCAD'13).* 165–168.

Edmund Clarke, Orna Grumberg, and Doron A. Peled. 2001. *Model Checking.* MIT Press.

Edmund Clarke, Bruce Krogh, Andre Platzer, and Raj Rajkumar. 2008. Analysis and verification challenges for cyber-physical transportation systems. In *Proceedings of the National Workshop for Research on High-Confidence Transportation Cyber-Physical Systems: Automotive, Aviation and Rail.*

Jason Croft, Ratul Mahajan, Matthew Caesar, and Madan Musuvathi. 2015. Systematically exploring the behavior of control programs. In *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC'15).* 165–176.

Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08).* 337–340.

Andreas Dolzmann. 2006. Redlog. Retrieved from http://redlog.eu/.

Google Brillo. 2016. Google Brillo. Retrieved from https://developers.google.com/brillo/.

Goran Frehse, Sumit Kumar Jha, and Bruce H. Krogh. 2008. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control (HSCC'08).* 187–200.

Ştefan Gună, Luca Mottola, and Gian Pietro Picco. 2014. DICE: Monitoring global invariants with wireless sensor networks. *ACM Trans. Sen. Netw.* 10, 4 (2014), 54:1–54:34.

Thomas A. Henzinger. 1996. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science.* 278–292.

Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. 1998. What's decidable about hybrid automata?*Journal of Computer Systems Science* 57, 1 (1998), 94–124.

Thomas A. Henzinger and Howard Wong-Toi. 1995. Using hytech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control*, vol. 1165. Lecture Notes in Computer Science, Springer, 265–282.

Douglas Herbert, Vinaitheerthan Sundaram, Yung-Hsiang Lu, Saurabh Bagchi, and Zhiyuan Li. 2007. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. *TAAS* 2, 3 (2007), 8:1–8:23.

IFTTT. 2011. IFTTT: Put the internet to work for you. Retrieved from http://ifttt.com.

Json.NET. 2009. Json.Net. Retrieved from https://www.newtonsoft.com/json.

Edward Lee. 2006. Cyber- physical systems- are computing foundations adequate?*Position Paper for NSF Workshop on Cyber-Physical Systems: Research Motivation, Techniques and Roadmap.* Austin, Texas. https://ptolemy.berkeley.edu/publications/papers/06/CPSPositionPaper/Lee_CPS_PositionPaper.pdf.

Xuandong Li, Sumit Jha, and Lei Bu. 2007. Towards an efficient path-oriented tool for bounded reachability analysis of linear hybrid systems using linear programming. *Electrical Notes on Theoretical Computer Science* 174, 3 (2007), 57–70.

Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Börje Karlsson, Dongmei Zhang, and Feng Zhao. 2016. Systematically debugging IoT control system correctness for building automation. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys@SenSys'16).* ACM, 133–142.

Chieh-Jan Mike Liang, Börje F. Karlsson, Nicholas D. Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. 2015. SIFT: Building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks (IPSN'15).* 298–309.

Shan Lin, Tian He, and John A. Stankovic. 2008. CPS-IP: Cyber physical systems interconnection protocol. *SIGBED Review* 5, 1 (2008), 22.

David Monniaux. 2008. A quantifier elimination algorithm for linear real arithmetic. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08).* 243–257.

Sirajum Munir and John A. Stankovic. 2014. DepSys: Dependency aware integration of cyber-physical systems for smart homes. In *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'14)*. Berlin, Germany, 127–138.

Nielsen Norman Group. 1993. Response Times: The 3 Important Limits. Retrieved from https://www.nngroup.com/articles/response-times-3-important-limits.

Stefan Schupp, Erika Ábrahám, Xin Chen, Ibtissem Ben Makhlouf, Goran Frehse, Sriram Sankaranarayanan, and Stefan Kowalewski. 2015. Current challenges in the verification of hybrid systems. In *Proceedings of the 5th International Workshop on Cyber Physical Systems. Design, Modeling, and Evaluation (CyPhy'15)*. 8–24.

Universal Devices Products. 2007. Universaldevicesproducts/insteon/isy-99iseries. Retrieved from http://www.universal-devices.com/.

Dingbao Xie, Lei Bu, Jianhua Zhao, and Xuandong Li. 2014. SAT-LP-IIS Joint-directed path-oriented bounded reachability analysis of linear hybrid automata. *Formal Methods in System Design* 45, 1 (2014), 42–62.

Received July 2016; revised June 2017; accepted January 2018