SAFETAP: An Efficient Incremental Analyzer for Trigger-Action Programs

McKenna McCall*, Faysal Hossain Shezan†, Abhishek Bichhawat*, Camille Cobb*, Limin Jia*, Yuan Tian†, Cooper Grace†, Mitchell Yang*
* Carnegie Mellon University, † University of Virginia

***Abstract*—Home automation rules that allow users to connect smart home devices using trigger-action programs (TAP) can interact in subtle and unexpected ways. Determining whether these rules are free of undesirable behavior is challenging; so researchers have developed tools to analyze rules and assist users. However, it is unclear whether users need such tools, and what help they need from such tools. To answer this question, we performed a user study where half of the participants were given our custom analysis tool SAFETAP and the other half were not. We found that users are not good at finding issues in their TAP rules, despite perceiving such tasks as easy.**

The user study also indicates that users would like to check their rules every time they make rule changes. Therefore, we designed a novel incremental symbolic model checking (SMC) algorithm, which extends the basic SMC algorithm of SAFETAP. SAFETAP$^{\triangle}$ only performs analysis caused by the addition or removal of rules and reports only new violations that have not already been reported to the user. We evaluate the performance of SAFETAP$^{\triangle}$ and show that incremental checking on average improves the performance by 6X when adding new rules.

I. INTRODUCTION

Automatic interaction among home devices is making the Internet of things (IoT) platforms more user friendly and popular. Some of the most popular platforms are IFTTT [29], SmartThings [44], openHAB [39], Microsoft Power Automate [36], Zapier [53], and Homekit [4]. Generally, IoT platforms follow the *trigger-action programming* (TAP) paradigm to create automation *rules* which *trigger* based on some condition and perform an *action* as a result. For instance, users can install the rule "IF Nest Protect detects carbon monoxide THEN turn Phillips Hue light red" that turns the light red (action) to indicate that carbon monoxide is detected by Nest Protect (trigger). Currently, more than 11 million people use IoT automation rules in their home devices [30].

TAP automation rules connect IoT devices and can interact with each other, without the involvement of the user. Unfortunately, this may cause devices to behave differently than the user intends because the devices can interact in subtle and unexpected ways. Prior work has shown that interactions between rules and devices have the potential to create undesirable and unsafe states in smart home environments (e.g., leaving doors unlocked when the user is not home [12], [13], causing users' private information to be leaked [11], or allowing untrusted parties to control users' home devices [45], [55]).

As an illustration of unexpected behavior caused by rule-interactions, consider a user who installs the following rules:
(R1) "IF my garage is opened THEN send me a notification"
(R2) "IF I am at work THEN turn off my notifications"
When the user is at work, she won't be notified right away if her garage is opened because of the rule (R2). Interactions between rules—rules triggering other rules, rules interacting via the physical environment, multiple rules interacting with the same device, or even triggering the same events—make it difficult to guarantee that the rules, and by extension, the devices, will behave as intended.

Studies have shown that users have trouble understanding interactions between multiple rules or interpreting actions and triggers of rules in IoT platforms; Brackenbury *et al.* [9] showed that users have trouble predicting the behavior of TAP rules when the rules have bugs. Researchers have proposed tools for identifying such unsafe behavior in TAP rules (and IoT applications) by using model checking [12], [28], [32], [38], [48], [55]. Some of this work focus on describing general unsafe behaviors that rule-interactions can cause [32], [38], [48]. Other works [12], [28], [55] have designed more targeted tools for users to specify their desired behaviors which are checked for against the user-installed rules and mostly are limited to checking safety properties, e.g., temperature of the house should never be more than 75°F[1]. It is unclear whether users need such tools and to what extent such tools can help users in identifying unsafe behavior in their TAP rules.

Based on these observations, the first goal of our work is to answer these questions via a user study. To do so, we designed and implemented a tool SAFETAP that allows users to check for different types of desired behaviors. We forego general-purpose model checkers like SPIN [27], Uppaal [31] or NuSMV [16], and instead develop a symbolic model checking tool for TAP rules. By focusing only on TAP rules, we can apply efficient domain-specific strategies.

We designed a user study to evaluate how well users understand the nuances of TAP rule interactions and to determine if they can identify problems themselves, or if they need a tool like SAFETAP to assist them. We gave half of the participants access to a web interface for SAFETAP to additionally measure whether SAFETAP helped them find problems. We designed various tasks where they were asked to determine if a set of rules would behave as intended by a pretend user. We randomly assigned the participants tasks to perform either for "Alice" who had 8 rules installed or for "Bob" who had 5 rules installed, to see whether and how the number of rules would affect their performance.

In general, people were able to identify when the rules *would* behave as intended, but they performed about *half* as badly at spotting problems, despite most of the participants

[1]The terms "behaviors" and "properties" are interchangeable in this paper

reporting that they found the tasks “easy”. Moreover, participants were approximately *five times more likely* to spot problems with the “Bob” ruleset than with the “Alice” ruleset, even though “Alice” had only 3 more rules none of which contributed to any misbehavior. Users tended to perform *better* if they were given the SAFETAP web interface, even though only a small number of participants used it to check behaviors. Our findings suggest that users need tools to help them understand how TAP rules interact and, more specifically, to help them spot problems and what cause them.

An important observation from the user study was that about 57% of the participants indicated that they would be interested in using such a service every time they installed or modified a rule. Our study also led us to hypothesize that participants struggled with finding problems because whenever they found a rule which did what they expected, they assumed that no other rules would interfere. Consequently, we believe that they would struggle *even more* with the task of determining if a ruleset will behave as they expect *after* adding a new rule or removing an existing rule, if the rule does something desirable. These observations motivated the second goal of this work: improve model checking-based analysis to be more efficient and responsive to users’ incremental changes to their TAP rules. More specifically, we aim to provide users with a tool to easily verify and *incrementally* re-check their rules as they make changes.

Our incremental analysis works by storing state from previous analyses and using it in future analysis so that only the modifications in the system are re-analyzed, thereby avoiding the repetition of previously performed analysis. Newly-added rules can be incrementally analyzed in combination with all other rules without having to check for interactions amongst the existing rules. This has two advantages—a reduced cost for checking the behavior, and not flagging the user repeatedly for violations that they have already ignored. This is particularly useful when small changes are made to a large rule set. Similarly, removal of rules only requires removal of the previously stored state related to those rules, and can identify the rule whose removal induced a violation, if any.

As SAFETAP is a light-weight custom tool, we have full control of the analysis algorithm and the internal state of the model checking. We developed a novel incremental symbolic model checking algorithm for TAP rules and integrated it with SAFETAP (which we call SAFETAP$^{\triangle}$). SAFETAP$^{\triangle}$ does not re-check already analyzed unchanged rules and only reports new issues caused by rule changes.

SAFETAP and SAFETAP$^{\triangle}$ are implemented in Python. Our evaluation of SAFETAP shows that it can detect all property violations for a set of 100 rules in less than 60ms and scales well. We evaluate SAFETAP$^{\triangle}$ by comparing its performance in subsequent analyses (after an initial analysis which stores the state for incremental analysis) with SAFETAP, and find that SAFETAP$^{\triangle}$ is on an average 6 times faster than SAFETAP when performing incremental analysis.

This paper makes the following contributions:

- A web-based tool, SAFETAP, to help users check for unsafe behavior of their TAP rules.
- A user study assessing users’ understanding of TAP rules and needs for tool assistance.
- An efficient incremental analysis tool, SAFETAP$^{\triangle}$, that only analyzes modified rules.

II. BACKGROUND

We briefly describe trigger-action programming (TAP) and review key concepts in model checking.

A. End-user trigger-action programming for home IoT

Typically, the home IoT ecosystem includes three main components: hardware, applications, and automation platforms. The hardware includes the IoT devices which have sensors that collect data from the physical environment or affect the physical environment. For instance, a location tracker might return the GPS coordinates for the device and a temperature sensor would return the current temperature of room in which it is installed.

An application processes the data collected by hardware. Based on the collected data, an application may control hardware devices, perform analytics and reporting, or communicate with the server and other devices. For instance, an application controlling the heater in a room may turn on the heater if the temperature of the room drops below a certain threshold, and turn it off once the temperature reaches a preset value.

Automation platforms like IFTTT [29] and SmartThings [44] allow users to programmatically form links between devices and applications and web services via *trigger-action programming* (TAP). Generally, TAP rules are specified in the format: “IF *trigger* THEN *action*”, where trigger is a state change reported by the devices (e.g., alarm is turned off) or state of the environment (e.g., the temperature is below 75°F), and action is the update to device status or other services (e.g., turn on the alarm, or send me an email).

When rules’ actions alter the environment, they may trigger other rules—intentionally or unintentionally.For instance, consider the rules: (R1) “IF the user nears home THEN unlock door” and (R2) “IF door is unlocked THEN start recording with camera”. Once the user reaches home, the rule (R1) is triggered, and the action (unlock door) triggers the rule (R2), causing the camera to start recording. This may lead to unexpected behavior because it can be hard to anticipate how and when the rules will interact.

B. Model checking

Model checking is an approach to determine if a formally-modeled system satisfies a specification, written as a set of properties. The model is an abstract formal representation of the system, which is given by a finite number of states and transitions between the states. A state is a snapshot of the system, which, in our case, includes variables and their respective values representing all the device statuses and the physical environments. Transitions from one state to another are based on events (triggers and actions) in the system. Properties are usually specified as formulas in temporal logics like linear temporal logic (LTL) or computational tree logic (CTL). The model is typically represented as a transition system with every node of the system representing a unique state [6].

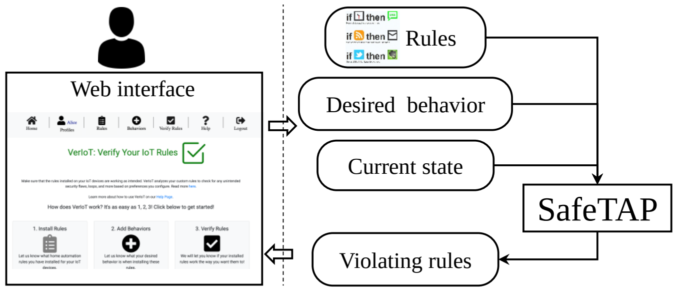


Fig. 1: System Architecture for Property Checking with SAFETAP

SAFETAP uses a fragment of CTL for specifying properties. The system's behavior is modeled as a computational tree. Along with standard logical operators, CTL formulas include temporal operators (X, G, F and U) quantified over paths in the tree, which are called traces. Figure 10 in Appendix B illustrates the semantics of these operators. The existential quantifier, written E, says that the property holds for at least one trace, while the universal quantifier, written A, says that it holds for every trace, starting from the initial state. A state formula, α, is a fact determined by looking at a single state.

The next operator, X α, says that the formula α holds in at least one (EX) or all (AX) states reachable from the current state in a single transition. The global operator, $\mathsf{AG}(\alpha)$ holds in the current state if α holds in all subsequent states. Similarly, $\mathsf{EG}(\alpha)$ says that α holds in all states along at least one trace. The future operator, $\mathsf{AF}(\alpha)$, says that, regardless of which transitions we make, α will eventually hold in the future, while the property $\mathsf{EF}(\alpha)$ says that α will hold along some path at some point in the future. The until operator, $\alpha \ \mathsf{U} \ \alpha'$, says that α is true at least until we reach a state where α' is true, in at least one (EU) or all (AU) paths starting from the initial state.

Instead of traversing one reachable state at a time, a more efficient approach to CTL model checking is *symbolic model checking* (SMC) [17], [35] that considers a set of states in a single step by using a symbolic representation for those states. We describe the algorithms for SMC used by SAFETAP in Section VI and Appendix D.

III. SAFETAP ARCHITECTURE AND DESIGN

An overview of the system architecture with SAFETAP, an analysis tool for checking whether a set of TAP rules satisfy desired properties, is shown in Figure 1. SAFETAP takes as inputs: TAP rules, desired behaviors (or properties) to be checked (see Section IV-A for a list of properties that SAFETAP can check), and the current state of the system. When the rules do not satisfy a property, SAFETAP generates all violating cases, composed of sequences of violating rules, as output. We have built a web interface for users to interact with SAFETAP (Section IV-B).

Next, we provide an overview of SAFETAP, high-level descriptions of how trigger-action programming platforms are modeled in SAFETAP and the algorithm behind SAFETAP via examples. Finally, we present motivating scenarios for SAFETAP$^{\Delta}$ in Section III-B, an extension of SAFETAP that implements our novel incremental analysis.

A. SAFETAP *Design*

SAFETAP is a custom analyzer which builds on classic SMC algorithms for checking TAP rules for any violations of the properties. Building a custom tool allows us to easily report *all* violations at once, consider the impact of users' actions (e.g., entering the house), and extend it to incrementally analyze the rules in real time (in more detail in Section III-B).

As is standard in SMC, SAFETAP needs a formal specification language for the properties and a model of TAP. Here, we describe the syntax necessary for modeling TAP, give an informal overview of the model, and explain the algorithm using examples.

Formulas used by SAFETAP. SAFETAP properties are written in a fragment of CTL, where we restrict the nesting of temporal operators to what is required for the properties in Section IV-A. For checking properties in the universally-quantified fragment (e.g., AG and AU), SAFETAP looks for violations of the property by translating it to the existentially-quantified fragment and checking the satisfiability of the negated formula, as is standard in SMC [6], [17], [35].

Consider the scenario from Section I where the user wants to be notified whenever her garage door is opened. The property is specified in CTL as:

$$\mathsf{AG}(\text{garage door opened} \rightarrow \text{notify user})$$

The equivalent formula using only existential operators is:

$$\neg \mathsf{E}(\top \ \mathsf{U} \ (\text{garage door opened} \wedge (\neg \text{ notify user})))$$

This property says that there does not exist a trace that reaches a state where the garage door is opened and the user is not notified. SAFETAP checks whether there *exists* a trace that reaches this "bad" state, i.e.,

$$\mathsf{E}(\top \ \mathsf{U} \ (\text{garage door opened} \wedge (\neg \text{ notify user})))$$

If SAFETAP finds traces satisfying this property, a violation is found and reported to the user; otherwise, SAFETAP reports that no violations are found.

The properties in Section IV-A are specified over all paths. We translate these to existential properties, like in the example above; the satisfaction of the translated property means there *is* a violation. SAFETAP returns all traces which satisfy these existential properties (i.e. traces which are *counterexamples* for the original, universally-quantified properties).

Modeling the trigger-action platform. TAP rules are modeled as a labeled-transition system. Given the current state of the environment and some event, for instance "temperature rising", the system checks to see if any TAP rules are triggered in Step 1. Next, the system collects all of the triggered rules in Step 2 by comparing the triggers to the event and the current environment. The rules trigger actions in Step 3, which may themselves be events, or state effects. Finally, in Step 4, the state is updated to reflect the state effects and the system returns to Step 1 to process any events which were triggered in Step 3. The complete syntax, figure and formal semantics can be found in Appendix A.

We model interactions between the rules as a computation tree, which represents all possible executions starting from a given initial configuration, represented as the root of the tree. Each node in the tree is a state in the system's execution. Each edge from a parent node to one of its children represents a possible transition (triggered by an event). Since it is possible for multiple events to trigger rules in any particular state, a node may have multiple children.

Example safety property analysis in SAFETAP. Our algorithms use formulas to represent a set of states and work backwards. We start at the set of states which satisfy the property and compute a pre-state formula to compute the set of conditions necessary to reach these states. The pre-state formula lets us walk backward through the tree until we reach a fixed-point. Our algorithm then checks if the initial state is in the set of states that satisfy the pre-state formula. If it is, we know the property is satisfied. If it is not, we know that the property does not hold.

To understand how SAFETAP analyzes properties, consider the example described earlier where the user specified the property: "Whenever the garage door is opened, I should be notified" and installed the rules:
(R1) "IF my garage is opened THEN send me a notification"
(R2) "IF I am at work THEN turn off my notifications"
The property, which we input to SAFETAP is:

$$\mathsf{E}(\top\ \mathsf{U}\ (\mathit{garage} = \mathsf{Open} \wedge \mathit{notify} = \mathsf{False}))$$

Suppose in the initial configuration the garage door is closed, and the notifications to the user are turned on, i.e, ($\mathit{garage} = \mathsf{Closed} \wedge \mathit{notify} = \mathsf{True}$).

SAFETAP begins with the set of states which satisfy the condition: ($\mathit{garage} = \mathsf{Open} \wedge \mathit{notify} = \mathsf{False}$). First, it checks if the set of states satisfying this condition contains the initial conditions (garage door is closed and the notifications to the user are turned on). Since they do not, it then generates the pre-state formula by checking the actions of the two rules. When traversing the action of (R2), it updates the action in the condition and adds the trigger to the formula, i.e., the pre-state formula computed by SAFETAP is ($\mathit{garage} = \mathsf{Open} \wedge \mathit{location} = \mathsf{Work} \wedge \mathsf{False} = \mathsf{False}$). As the event, "opening the garage door", can be a user-triggered event, SAFETAP simulates the action of this user-event and generates a new pre-state formula where *garage* is set to Open, i.e., the pre-state formula is $\mathit{location} = \mathsf{Work}$. As no more rules can be traversed on this path, it returns this as the final disjunction of the pre-state formula. As the initial conditions satisfy this formula, SAFETAP reports a violation due to rule (R2).

B. Incremental Analysis

To provide feedback to users, we extend SAFETAP to *incrementally* analyze rules for violations of behaviors, i.e., when new rules are added (or removed) by the user. In this case, the algorithm verifies only the interactions induced by the newer rules for property violation. More specifically, SAFETAP$^{\Delta}$ stores partial results from previous analysis of a behavior and when a new rule is added or an existing rule is removed, it restricts its analysis to the set of rules that was added or removed; thus, it avoids having to analyze the entire ruleset every time it is modified.

Incrementally analyzing new rules We illustrate the incremental analysis with a smart-home example [13]. Consider a user who has installed the following three rules:
(R1) When I enter home, turn the lights on,
(R2) When the lights are turned on, activate home-mode,
(R3) When home-mode is activated, turn on the heater
Suppose that the user wants: "Whenever I exit home, make sure that heater remains turned off". None of the three installed rules or their interactions violate this behavior.

Now, suppose that the user is going on a vacation, and installs another rule (R4) to simulate occupancy by turning on and off the lights when a button is tapped in a mobile app. Traditional model checkers would have to verify all installed rules and possible interactions to check if the desired behavior is violated, repeating all of the analysis performed before. SAFETAP$^{\Delta}$, on the other hand, only checks if (R4) or combinations of rules involving (R4) violates the behavior, and reports this to the user.

Incremental analysis in the presence of prior violations. When adding a new rule, SAFETAP$^{\Delta}$ incrementally checks whether the rule or its interaction with all other rules violates the property. This has two advantages—a reduced cost for verifying the property and not notifying the user repeatedly for the same violation. The latter might be useful from the perspective of usability, as the user might not be interested in viewing a previously reported violation because they intentionally ignored it. Recall the example from Section I where a user has installed the following rules:
(R1) Turn off notifications when I am in a meeting
(R2) Send me a notification when the garage door is opened
and specifies the property—"Whenever the garage door is opened, I must receive a notification". The rule (R1) violates this property, which is reported to the user. The user might be fine with this violation as they might prioritize not wanting to be disturbed during a meeting. Suppose they install a new rule:
(R3) Turn off notifications at night
When (R3) is added, SAFETAP reports (R3) as the only rule that violates the behavior because the violation caused by (R1) was already reported.

Incrementally analyzing rules after removal. Similarly, incrementally verifying the removal of rules does not require all state formulas to be re-computed. Instead, SAFETAP$^{\Delta}$ uses the existing state formulas to decide whether the removal of a rule caused a new violation to occur and can identify the rule which caused a violation upon removal. For instance, consider a scenario where a user has installed two rules:
(R1) Turn off the air-conditioner in the morning
(R2) Turn on the air-conditioner at night
and the behavior—"Check that turn on A/C is always followed by turn off A/C" (see Section VIII for more details on this property). Rules (R1) and (R2) together satisfy this property but if the user removes rule (R1), the property is violated. SAFETAP$^{\Delta}$ can use the results from previous verification to report that the removal of the rule (R2) causes this violation without having to run the verification algorithm again.

IV. BEHAVIOR-SPECIFICATION INTERFACE

SAFETAP can be used to check any property that can be specified in the fragment of CTL described in Section III-A. To make the tool more user-friendly, we created a web interfacefor specifying and checking properties (which we call *desired behaviors* on the website). In this section, we describe the properties supported on the behavior-specification interface and introduce the web tool and its capabilities.

A. *Properties*

We name and describe each property the SafeTAP web tool supports. The CTL formulas used to check each property can be found in Table 9 in Appendix B.

Property 1 (Always/Never)**.** Always and Never properties (otherwise known as safety properties) capture high-level user intent and ensure that the system never enters a "bad" state. Safety properties are similar to some policies described in prior work [10], [12], [13], [32], [33], [38], [55].

Depending on the circumstance, it might be more natural to use Always or Never. For instance, a user might *always* want the temperature inside their house to be between $68°$ and $75°$. There is a matching Never property which expresses the same thing, but it is not as elegant: the temperature should *never* be below $68°$ or above $75°$. We allow the user to specify both Always and Never properties for convenience.

Property 2 (Whenever)**.** Whenever properties have the form, "*whenever* A happens, make sure that B happens". A property of this form is satisfied by checking that for all states where A is true, B also holds. Like Property 1, Whenever properties are useful for specifying at a high level how TAP rules should behave. For instance, a user might install a TAP rule to ensure that *whenever* they leave their home, their air conditioner is turned off, and want to verify that no other rule will interfere with this behavior.

One noteworthy use for this property was exemplified in Section I where the user installs the rule "IF my garage door is opened THEN send me a notification.". The effect of this rule was overridden by another rule that mutes the notifications while the user is at work. The property, "*whenever* my garage door is opened, make sure that my notifications are on" would identify the rule that mutes notifications as problematic.

Property 3 (Only When)**.** Sometimes it is not enough to ensure that A happens whenever B is true. The user may also want to know that A happens *only when* B is true. This is especially useful for rules which are supposed to signal to the user when something important happens. Consider the scenario where a user installs the following rule: "IF smoke detected THEN blink light". The intention here is to be notified when smoke is detected by a blinking light. If this user forgets that they also have the rule "IF World Cup update THEN blink light" installed, this could be a problem. When the user's light blinks, they might not know without further investigation whether there is a game update or there is a fire in their house.

To ensure that there is no ambiguity about which event (smoke or the World Cup) triggers a signal (a blinking light), the user might want to check that the important action occurs *only* after a particular trigger.

Property 4 (No Loops)**.** TAP allows rules to be chained together to perform complex tasks. However, chaining can have unintended consequences if the chains contain loops.

For example, the rules "IF an event is added to your calendar THEN add a reminder" and "IF a new reminder is added THEN create an event on your calendar" creates a loop. Triggering either of these rules triggers the other, creating infinitely many reminders, and calendar events. This property is also identified as undesirable by Wang et al. [48] and is similar to the "loop triggering" property identified by Chi et al. [15], which accounts for more subtle chaining due to environmental changes, like we do.

Property 5 (No Action Cancelling)**.** Action cancelling (also referred to as action conflict [12], [13], [33], [38], [48]) occurs when the triggers of two rules overlap, but their actions disagree. For example, the rules: "IF user leaves home THEN turn on security camera" and "IF time is 6AM THEN turn off security camera" cancel each other's actions. If the triggers (user leaves home, and time is 6AM) happen simultaneously (user leaves home at 6AM), then both rules will be activated.

These rules disagree on what to do; while one rule turns on the security camera, the other one turns off the security camera. Because TAP rules are not usually installed with priorities to determine what order they are processed, it is hard to predict what will happen when two rules cancel each other's actions. Whichever rule happens to be processed first will have its action cancelled by the rule processed second, which could lead to unexpected behavior.

B. SafeTAP *Web Interface*

The SafeTAP web interface [2] (shown in Figure 11 in Appendix C) helps users write and check properties, also called *desired behaviors*, for their TAP rules. The SafeTAP web interface also supports profiles. For our user study, profiles were used to simulate different "users". This feature is also useful for users with IoT devices that do not interact with each other (for instance, some devices might be at their homes, while others are at their office) to check each setting "home" and "office" independently. Users can check that their TAP rules will behave the way they intend in 3 steps, which we outline below.

1) Install Rules: The first step is to install rules, which are organized by service. For example, the rule "Turn on camera if the user is away from home" appears under the "Location" and "Camera" services. Currently, the set of rules available to install are a small subset of IFTTT rules. Rules can also be temporarily deactivated.

2) Add Behaviors: The next step is adding properties. To do so, the user clicks on a behavior type and fills out a template.

Templates are available for the Always ("I always want [condition]"), Never ("I never want [condition]"), Whenever ("Whenever [condition 1] make sure [condition 2]"), and Only When ("[condition 1] only when [condition 2]") behavior types. Loops (Property 4) and action cancelling (Property 5) do not involve conditions, so they do not require templates.

[2]When we released the website, our tool was named VerIoT, but we changed it to SafeTAP as Yuan et al. [52] proposed a tool with a similar name.

3) Verify Rules: The final step is to run SafeTAP to check if the TAP rules selected by the user will behave as desired. The verification page includes a verification summary of all of the behaviors at the top of the page, as well as the verification status of the configured desired behaviors. The summary includes buttons for checking Property 4 for rules which create loops and Property 5 for rules which cancel each other's actions.

To run SafeTAP, the user simply clicks the checkmark next to the behaviors they are interested in checking. Green checkmarks mean the TAP rules they selected satisfy that property. Red X's mean that violations were found. Clicking the red X displays which rule(s) were involved in the counterexample violating the property.

V. User Study

We performed a user study to measure how well users can evaluate whether a set of rules will satisfy a property (or desired behavior). The goal of our study was to determine whether they can identify problems themselves, or if they require the help of a tool like SafeTAP to assist them. In the process, we evaluated the usability of the web interface for SafeTAP.In this section, we discuss how we designed our user study. In addition to describing the high-level methodology, we outline the types of questions that appear in the survey we distributed as well as the tasks we gave the participants[3]. Finally, we discuss the results of our study and its limitations.

A. Methodology

We initially recruited 55 people through Amazon Mechanical Turk [1] to participate in our study. We required that participants have at least an 85% HIT acceptance rate on previously-completed MTurk tasks, be able to speak and read English fluently, be over the age of 18, and "have prior experience with IoT devices." 18 responses were not included in our analysis because these participants either did not follow our directions or their answers conveyed that they did not pay attention.We were left with 37 "valid" responses. Unfortunately, we found that very few people used the website (we discuss this in more detail in Section V-E), so we used only the responses without access to SafeTAP from this first round. We re-released the study and additionally required that people use the website to perform the tasks and collected 40 additional responses. Of these, 22 were "valid" responses, by the same metrics listed above, plus participants were also excluded if we knew they did not use the website because they did not enter a valid website code (generated by the web tool) in the survey.

The study was approved by UVA's IRB. We paid participants a base rate of US$5, plus up to a US$3 bonus, as described below. The survey consisted of 4 sections:

1) **Experience with IoT:** Participants specified what type of experience they had with IoT (e.g., if they own IoT devices, how often they interact with IoT devices, and whether they use TAP systems).
2) **Rule Evaluation Questions:** Section V-B
3) **Survey Follow-up:** Participants answered questions about the survey tasks, as explained below.
4) **Demographics:** The study concluded with a set of demographic questions.

[3]The full survey questions may be found at https://safetap.s3.amazonaws.com/userstudy.pdf

B. Rule Evaluation Questions

Number of participants			
	Without SafeTAP (-)	With SafeTAP (+)	Total
Alice Task (A)	9	8	17
Bob Task (B)	9	14	23
Total	18	22	40

Fig. 2: Distribution of participants each survey condition: Alice task without SafeTAP (A-), Bob task without SafeTAP (B-), Alice task with SafeTAP (A+), Bob task with SafeTAP (B+)

The primary goal of the rule evaluation section is to assess whether participants are able to determine if a set of rules satisfy a given property (called a "desired behavior" in the survey). We varied the number of rules to measure whether the difficulty of the tasks scales with the number of TAP rules. We varied the inclusion of SafeTAP to evaluate whether it was helpful to the participants, and to measure the usability of the web interface. The distribution of participants, after removing "invalid" responses, is shown in Figure 2.

Each participant received two tasks with different *rule sets* and three questions for each task. The rule sets were static throughout the task. Each question had a *desired behavior specification* (property) and the participant was asked to decide if the rule set would satisfy it. Regardless of whether they gave the expected correct answer or not, participants were asked to indicate which rules support their answer. That is, if they say that the behavior *is* satisfied, we asked them which rules must remain installed to achieve this behavior? And if they say that the behavior is *not* satisfied, we asked them which rules should be uninstalled to fix the problem?

Before asking the participants to begin the tasks, we gave them an example rule set and shared the expected correct answer with participants so they knew what to expect. The main survey tasks asked participants to consider two sets of TAP rules installed by "Alice" (or "Bob", depending on which group they were in) and "Charlie." Alice and Bob's set of desired behavior specifications were the same and they also had the same correct answers even though Alice's rule set had eight rules while Bob's had only five. Alice and Bob's rule sets were the same except for the three additional rules in Alice's rule set, which were not required to justify any of the correct answers (although they might be related to the properties we asked about).

We group questions based on how they can be answered with SafeTAP. The questions for Alice and Bob's sections could be answered using Never or Whenever properties. We group these questions together because these properties all specify high-level user intent and could be expressed more than one way, depending on which property better fit their intuition. The questions for Charlie's section cannot easily be expressed multiple ways and require more specific properties. The first is meant to be checked with an Only When property, the second checks if there are No Loops, and the third asks if there is No Action Cancelling.

To discourage participants from randomly guessing, we offered a US$0.50 bonus for each question that was answered correctly *or* accompanied by a thoughtful free-response answer (i.e., up to US$3 bonus, or US$8 total compensation).

SAFETAP (+) Condition The survey for the (+) condition is very similar to the (-) condition. It includes a modified set of instructions, specific to the SAFETAP web interface, with a tutorial video as well as follow-up questions about their experiences with SAFETAP.

To keep the effort similar among all participants, we pre-loaded profiles for Alice, Bob, and Charlie with the rules for each section already loaded. We did not allow them to add, delete, or edit the rules on these profiles. To complete the tasks, the participants only needed to add the properties, run the verification, and interpret the results. At the end of the tasks, they were encouraged to create their own profiles and explore the site with full functionality, including adding their own rules.

C. Survey Follow-up

After they completed the rule evaluation questions, participants in the (+) condition answered a series of questions about their experiences with SAFETAP, such as the difficulty of adding a desired behavior in SAFETAP and whether they found SAFETAP useful. This helped us evaluate the usability of SAFETAP. Participants in all conditions indicated their perception of the difficulty of answering the questions on a Likert scale. We were interested whether people seemed to overestimate their ability to understand TAP rule interaction.

At the end of the survey, participants were invited to give "any other feedback" they wanted to share.

D. Survey Data Analysis

Before publishing the survey, we created an "answer key" for the tasks. Each question includes a Yes/No answer and a set of rules supporting that answer (when applicable). We considered two correctness measures: both Yes/No answer *and* rules match our answer key, and Yes/No answer matches our answer key. We were strict with our scoring and did not take into account any explanations in the free-response boxes for each question when deciding if the answers were correct. This is because many participants did not take advantage of the free-response boxes to explain their answers, so there was no principled way to determine if "incorrect" answers should be considered correct.

E. Results of user study

In this section, we highlight the key findings of our user study. General statistics about IoT and home automation experience and demographic data are summarized in Appendix E.

Users struggle to identify problems with TAP rules. Participants' average score for the tasks was 70% (4.18 of 6 questions answered correctly). This is consistent with results from prior work that show that users do not fully understand TAP rule interactions, especially in the presence of bugs [9]. Even more troubling, participants were more successful at identifying when behaviors *would* be achieved by a set of rules than when it *would not be* (90% vs 60% correct on average).

Further, even when they correctly determined *whether* the rules caused a problem, participants were only able to identify the rules that contributed to this problem 39% of the time. This statistic drops to 25% of the time if we exclude the tasks where they checked for loops and action cancelling, which people were generally more sucessful at[4]. In a real-world scenario, it would be important to identify not only when there is a problem, but precisely which rule causes the problem.

Despite the relatively poor performance when identifying problems, 70% of participants expressed that they agreed or strongly agreed that the tasks were easy. As justification for an incorrect answer, P14 wrote: "[rule] is obviously correct." Several other participants gave similar justifications for their incorrect answers. Explanations like this illustrate not only that participants are overly confident in their abilities to understand how TAP rules behave, it also gives us some insight into why they might have answered incorrectly: their strategy may have been to skim the list of rules only looking for a rule which produced the expected behavior, not realizing that *other* rules interfered.

Our study also gave us some insight into the types of properties users struggle with. Of the four questions in the survey where participants should have identified a problem with the rule set, two were Whenever/Never behaviors, and two were looking for loops and action cancelling. We found that participants scored much worse when looking for problems with the Whenever/Never behaviors (35% compared to 84% average score for identifying loops and action cancelling).

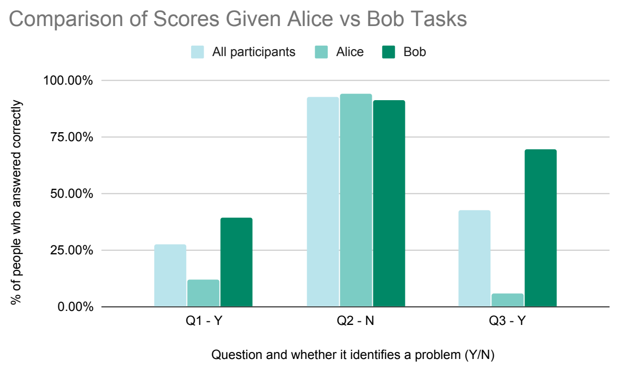


Fig. 3: Average scores for all participants with Alice vs Bob task

Users struggled more with more TAP rules. The Alice ruleset (8 rules) is the same as the Bob ruleset (5 rules) except that it has 3 additional rules, which do not directly contribute to any of the problems. The Alice vs. Bob condition was designed to test the hypothesis that a larger rule set made the questions more difficult to answer correctly. Our results support this hypothesis: participants with the Alice ruleset

[4]80% correctly identified that there was a loop and 88% correctly identified that there was action cancelling, but sometimes people only identified one of the rules involved in the interaction instead of both rules

correctly identified that there was a problem just 9% of the time, as opposed to the people with the Bob ruleset who correctly found a problem 55% of the time. It is not surprising that the difficulty of the tasks scale with the number of rules, but it is surprising how much of a difference only 3 rules made. This is an especially startling result given that 5 of the 22 participants who use TAP themselves reported having 10 or more rules, and prior work found that IFTTT users on online home automation forums had on average 26 rules installed (and as many as 66 rules) [18].

SAFETAP helped people perform (marginally) better. The results thus far support the claim that a tool such as SAFETAP could help users identify problems with their TAP rules as well as how to fix them. Unfortunately, participants in our study performed only marginally better at identifying problems when they had access to SAFETAP: 43% correctly identified the violation using SAFETAP compared to 32% without.

Of course, the usability of SAFETAP may have been a contributing factor; however, the feedback we received about the participants' experiences with the tool was fairly positive. 82% of the participants that had the opportunity to interact with SAFETAP agreed or strongly agreed that "it was easy to add desired behaviors for the tasks". Similarly, only 14% of participants disagreed with the statement that it was "easy to understand the results" after running SAFETAP.

The free-response justifications for the questions point to other factors that may have contributed to the finding that SAFETAP helped only a small amount. Participants' overconfidence and belief that the tasks were easy may have led them to believe that they did not need to use SAFETAP. This is supported by our analysis of SAFETAP logs. We could confirm the activity of only 11 participants (50% of those with access to SAFETAP). These are the people we know used the website to install and verify behaviors, but we are not sure to what extent they used the website to complete the tasks, nor can we be sure whether or not the other 11 participants used SAFETAP. Given the uncertainty about who used SAFETAP, we cannot draw any meaningful conclusions about whether or not using SAFETAP helps users determine if TAP rules behave as intended.

Taken as a whole, these results suggest that participants struggle to identify TAP rule(s) that cause problems, especially for Whenever/Never behaviors, and they are overly confident in their ability to do so. Their difficulties increase quickly with larger rule sets. It is therefore reasonable to conclude that users would benefit from a tool that could help them identify unintended behaviors, especially as rules are added.

VI. INCREMENTAL ANALYSIS IN SAFETAP$^{\Delta}$

Users may install new rules when new services are added or when they buy new devices, and modify or delete old ones; they may do so based on feedback from SAFETAP. Reanalyzing all the rules every time changes are made would cause a perceptible slowdown in how long it takes SAFETAP to verify rules. We leverage the characteristics of TAP to implement a novel *incremental* analysis of the rules where SAFETAP$^{\Delta}$ stores the state from previous analysis, checking only newly added or modified rules.

Algorithm 1 Algorithm to incrementally verify $\mathsf{E}(\alpha \;\mathsf{U}\; \beta)$

```
1: function INCADDEU($state_0$, $\alpha$, $\beta$, $(\varphi, \mathcal{L}, R_{old})$, $R_{new}$)
2:     $\varphi = \varphi \lor \beta$
3:     $R_{new} = R_{new} \setminus (R_{new} \cap R_{old})$
4:     $R_{all} = R_{new} \cup R_{old}$
5:     if SATCONF($\varphi$, $state_0$) then
6:         ▷ A violation found in initial state
7:         return ()
8:     end if
9:     $\mathcal{L}' = \ell'_n = []$
10:    $\ell_n$ = PREDSTATELIST($[(noop, \beta, [])]$, $R_{new}$)
11:    if $\mathcal{L} \neq []$ then
12:        for all $\ell \in \mathcal{L}$ do
13:            $\ell' = \ell_n \mathbin{++} \ell'_n$
14:            for all $(ev_i, \varphi_i, R_i) \in \ell'$ do
15:                $\varphi = \varphi \lor (\varphi_i \land \alpha)$
16:                if SATCONF($(\varphi_i \land \alpha)$, $state_0$) then
17:                    ▷ A violation found
18:                end if
19:            end for
20:            $\ell'_n$ = PREDSTATELIST($\ell$, $R_{new}$)
21:            $\ell_n$ = PREDSTATELIST($\ell_n$, $R_{all}$)
22:            $\mathcal{L}' = \mathcal{L}' \mathbin{++} [\ell \mathbin{++} \ell']$
23:        end for
24:        $\ell_n = \ell_n \mathbin{++} \ell'_n$
25:    end if
26:    repeat
27:        $\varphi_o = \varphi$
28:        for all $(ev_i, \varphi_i, R_i) \in \ell_n$ do
29:            $\varphi = \varphi \lor (\varphi_i \land \alpha)$
30:            if SATCONF($(\varphi_i \land \alpha)$, $state_0$) then
31:                ▷ A violation found
32:            end if
33:        end for
34:        $\ell_n$ = PREDSTATELIST($\ell_n$, $R_{all}$)
35:        $\mathcal{L}' = \mathcal{L}' \mathbin{++} [\ell_n]$
36:    until $\varphi \to \varphi_o$ and $\varphi_o \to \varphi$
37:    return ($\varphi$, $\mathcal{L}'$, $R_{all}$)
38: end function
```

We describe the algorithm used by SAFETAP$^{\Delta}$ for incremental analysis in this section via an example. We show the algorithm for checking $\mathsf{E}(\alpha \;\mathsf{U}\; \beta)$, where α and β are two state formulas. We defer all other algorithms to Appendix D.

Example: Consider the scenario from Section III-B where the user desires the following behavior: "Whenever I am not at home, make sure that heater is not on" for the following rules (the third column shows a formula-based representation of the rule):

R1	When I enter home, turn the lights on	Location = Home ⇒ Lights = On
R2	When the lights are turned on, activate home-mode	Lights = On ⇒ Homemode = On
R3	When home-mode is activated, turn on the heater	Homemode = On ⇒ Heater = On

The violating property that SAFETAP$^{\Delta}$ looks for is:

$$P = \mathsf{E}(\top \;\mathsf{U}\; (\mathsf{Location} \neq \mathsf{Home} \land \mathsf{Heater} = \mathsf{On}))$$

It states that the bad behavior is a trace that has a state where user's location is not home and the heater is on.

The top-level function for incrementally verifying EU properties when new rules are added, INCADDEU, is shown in Algorithm 1. It takes as input the current state (state_0), formulas α and β, the state from previous analysis $(\varphi, \mathcal{L}, \mathsf{R}_{\mathsf{old}})$, and the newer set of rules, $\mathsf{R}_{\mathsf{new}}$, against which the formula $\mathsf{E}(\alpha \mathsf{\ U\ } \beta)$ needs to be checked. In the tuple $(\varphi, \mathcal{L}, \mathsf{R}_{\mathsf{old}})$, φ represents the final pre-state formula, $\mathcal{L}$ contains the list of states from every iteration, and $\mathsf{R}_{\mathsf{old}}$ are the rules involved from the previous analysis. If the tuple is $(\mathsf{False}, [], [])$, no analysis has been performed previously. INCADDEU returns the final pre-state formula (φ), an updated list of states $\mathcal{L}'$, which are used as input for the next analysis when users add or remove rules, and the set of rules $\mathsf{R}_{\mathsf{all}}$ involved in the current analysis.

In our example, R1, R2 and R3 do not violate the behavior. After the initial analysis, SAFETAP$^{\Delta}$ stores the state ($\mathcal{L}$) as follows: A., B. and C. are identifiers for the lists of formulas

A.	False	R1
	Location ≠ Home ∧ Lights = On ∧ Heater = On	R2
	Location ≠ Home ∧ Homemode = On	R3
B.	False	[R2, R1]
	Location ≠ Home ∧ Lights = On	[R3, R2]
C.	False	[R3, R2, R1]

(ℓ_n on line 28) generated in every iteration (lines 26-36 for the initial analysis) of the algorithm; the second column contains the state formula (φ_i on line 28) that was generated when processing the rule(s) mentioned in the third column (R_i on line 28). The events that lead to the pre-state formula (ev_i on line 28) like LOCATIONENTER for R1, LIGHTSON for R2 or HOMEMODEACTIVE for R3 are also stored but we omit them from the tables above for brevity. In the initial state state_0, Location ≠ Home, Lights = Off, Heater = Off and Homemode = Off.

The algorithm works by repeatedly computing the pre-condition for the given property and the set of rules until a fixpoint is reached. This corresponds to assuming that there is a state in the computation tree that satisfies β and moving up the tree to collect the states that satisfy α, and then checking if the state state_0 is in that collected set of states.

INCADDEU calls PREDSTATELIST (Algorithm 2) to generate the list of pre-state event, formula and rules involved in the pre-state formula tuples (shown as $(\mathsf{ev}_i, \varphi_i, \mathsf{R}_i)$). The function PREDSTATERULE takes as input an event, a formula and a rule representing a set of configurations and returns another event and a new formula. The goal of this function is to compute the precondition of reaching configurations satisfying φ, via rule r, which is triggered by event ev_t and generates event ev_a. The function UPDFORMULA takes the formula φ and modifies it according to the updates to the state variables to generate the updated formula. The function PREDSTATERULE conjuncts the pre-conditions of the rule α along with the updated formula to generate the formula φ'.

Here, the first row of A. is the formula representing the state that can reach the state that heater is on and user not at home by triggering R1, i.e., Location ≠ Home ∧ Heater = On with Location having the value Home; thus, Home ≠ Home ∧ Heater = On, which is False. The function PREDSTATELIST (Algorithm 2) updates the action in the formula (β = (Location ≠ Home ∧ Heater = On)) using UPDFORMULA and joins (∧) the trigger state to it. Intuitively, R1 is triggered when Location = Home but the property we are checking for requires Location ≠ Home, thus eliminating (False) R1 alone as a possible rule for satisfying P. Similarly, the second row of A. is the formula that can reach the state by triggering R2, and the third row by triggering R3.

Similarly, the first row of B. is the formula representing the state that can reach the state that heater is on and user not at home by triggering R2 and then R1, i.e., Location = Home ∧ Location ≠ Home ∧ Heater = On, which is False, and the second row by triggering R3 followed by R2.

The only row in C. is the formula that can reach the state that heater is on and user not at home by triggering R3 followed by R2 followed by R1, which is Location = Home ∧ Location ≠ Home or False.

The final fixed-point pre-state formula is the disjunction of all of the formulas in the second column:

$$\begin{aligned}\varphi = \big(&(\mathsf{Location} \neq \mathsf{Home} \wedge \mathsf{Lights} = \mathsf{On} \wedge \mathsf{Heater} = \mathsf{On})\\ &\vee (\mathsf{Location} \neq \mathsf{Home} \wedge \mathsf{Homemode} = \mathsf{On})\\ &\vee (\mathsf{Location} \neq \mathsf{Home} \wedge \mathsf{Lights} = \mathsf{On})\big)\end{aligned}$$

In the initial state state_0, Location ≠ Home. The initial state is in the set of states satisfying this formula only if either (Lights = On ∧ Heater = On) or (Homemode = On) or (Lights = On). As all of these three state changes are user-events triggered by the user when at home, the installed rules do not satisfy the property P. As the formula does not satisfy P, the desired behavior of the user is not violated.

Inserting New Rules. When the user installs the new rule (R4) to simulate occupancy by turning on and off the lights on a button tap (shown below), SAFETAP$^{\Delta}$ can incrementally verify the last installed rule using the state-table ($\mathcal{L}$) generated before. The initial state state_0 = [Location ≠ Home]. INCADDEU starts by testing the satisfiability of $\varphi \vee \beta$ in the initial state state_0 where φ is the disjunction of all state formulas in $\mathcal{L}$ and β = (Location ≠ Home ∧ Heater = On). The function SATCONF used in the algorithm is an abstract function used to check if the set of states satisfying the boolean formula $\varphi \vee \beta$ contains the state state_0. If it returns true, a violation is found in the initial state, i.e., the current state in the model already violates the condition being checked for. If the state in state_0 is not in the set of states satisfying $\varphi \vee \beta$, the algorithm computes the pre-state formula from the set of states satisfying β, i.e., given the set of rules $\mathsf{R}_{\mathsf{new}}$ that end up in states satisfying β, the algorithm searches for pre-states that satisfied α and the preconditions α_r for each rule in $\mathsf{R}_{\mathsf{new}}$.

R4	Simulate occupancy, when I am not at home	OccButton = On ⇒ Lights = On

After the rule R4 is added, the initial state that the user wants to check for contains Location ≠ Home and OccButton = On apart from the other state to simulate occupancy when the user is not at home. In the example, when R4 is added to the set of rules, SAFETAP$^{\Delta}$ adds another row to A.: (OccButton = On ∧ Location ≠ Home ∧ Heater = On), R4 and checks if this satisfies the property P. Further, it checks if either R1 or R2 or R3 can be triggered by R4, and hence adds another row to B.: (OccButton = On ∧ Location ≠ Home ∧ Heater = On), [R2, R4]. It checks if this formula satisfies P or not, and continues to the next iteration. It then adds another row to C.: (OccButton = On ∧ Location ≠

Algorithm 2 Auxiliary functions used by SAFETAP$^{\Delta}$

```
1: function PREDSTATERULE($\mathsf{ev}, \varphi, r$)
2:     $(\mathsf{ev}_t, \alpha_r) \Rightarrow (\mathsf{ev}_a, U, \mathcal{E}) \leftarrow r$
3:     if $\mathsf{ev} = \mathsf{noop} \lor \mathsf{ev}_a = \mathsf{ev}$ then
4:         $\varphi' = \alpha_r \land$ UPDFORMULA($\varphi, U, \mathcal{E}$)
5:         return $(\mathsf{ev}_t, \varphi')$
6:     end if
7:     return ()
8: end function
9:
10: function PREDSTATELIST($list$, R)
11:     $nlist = \mathsf{nil}$
12:     for all $(\mathsf{ev}_i, \varphi_i, \mathsf{R}_i) \in list$ do
13:         $nl = \mathsf{nil}$
14:         for all $r \in \mathsf{R}$ do
15:             $t =$ PREDSTATERULE($\mathsf{ev}_i, \varphi_i, r$)
16:             if $t \neq ()$ then
17:                 $nl = (t.\mathsf{ev}, t.\varphi, \mathsf{R}_i \mathbin{++} [r]) :: nl$
18:             end if
19:         end for
20:         $nlist = nl :: nlist$
21:     end for
22:     return $nlist$
23: end function
```

Home, [R3, R2, R4]. When it checks whether this formula satisfies P or not, it finds a violation as the user's location is not Home in the state state_0 and OccButton is On. Again, the fixed-point formula is a disjunction of all the pre-state formulas in the second column. The updated state-table is shown below with the newly added states in blue:

A.	False	R1
	Location $\neq$ Home $\land$ Lights $=$ On $\land$ Heater $=$ On	R2
	Location $\neq$ Home $\land$ Homemode $=$ On	R3
	OccButton $=$ On $\land$ Location $\neq$ Home $\land$ Heater $=$ On	R4
B.	False	[R2, R1]
	Location $\neq$ Home $\land$ Lights $=$ On	[R3, R2]
	OccButton $=$ On $\land$ Location $\neq$ Home $\land$ Heater $=$ On	[R2, R4]
C.	False	[R3, R2, R1]
	OccButton $=$ On $\land$ Location $\neq$ Home	[R3, R2, R4]

In the case when some prior analysis has been done, the algorithm proceeds by computing pre-states for only the rules in $\mathsf{R}_{\mathsf{new}}$ and their interactions amongst themselves and with the rules in $\mathsf{R}_{\mathsf{old}}$ until it reaches the point where the previous analysis stopped. It then continues with all the rules ($\mathsf{R}_{\mathsf{all}}$) to check if any further pre-state formulas are possible.

If no prior analysis has been performed, INCADDEU directly proceeds to line 26 and performs analysis of all rules in $\mathsf{R}_{\mathsf{new}}$ to determine the pre-state formulas. Once the function reaches a fixpoint, i.e., no more states can be added to the set of states satisfying the newly-generated formula φ, it returns φ. The new formula φ includes the pre-conditions of rules for different possible paths that could have led to β along with the boolean formula α. The formula φ corresponds to the maximal set of states that satisfy the original formula $\mathsf{E}(\alpha\ \mathsf{U}\ \beta)$.

Thus, instead of executing the complete functionality from scratch, SAFETAP$^{\Delta}$ is able to incrementally verify rules for behavior violations. Once SAFETAP$^{\Delta}$ reports the violation and the rules involved, the user can take appropriate actions.

In the incremental part of the algorithm (lines 11-36), the previously computed state formulas are only used to generate the pre-state formulas for interaction with the newer rules. The algorithm does not include them in the check for satisfiability (line 16), and hence, does not report violations from the previous analyses. Note that the final pre-state formula φ contains the actual formula, which is a disjunction of all the pre-state formulas from both previous and current analysis.

Algorithm 3 Algorithm to verify $\mathsf{E}(\alpha\ \mathsf{U}\ \beta)$ when rules are removed

```
1: function REMOVEEU($\mathsf{state}_0$, $\alpha$, $\beta$, $(\_, \mathcal{L}, \mathsf{R}_o)$, $\mathsf{R}_n$)
2:     if SATCONF($\beta$, $\mathsf{state}_0$) then
3:         ▷ A violation found in initial state
4:         return ()
5:     end if
6:     if $\mathcal{L} \neq []$ then
7:         $\varphi = \mathsf{False}$
8:         $\mathcal{L}' = []$
9:         for all $\ell \in \mathcal{L}$ do
10:            $\ell_n = []$
11:            for all $(\mathsf{ev}_i, \varphi_i, \mathsf{R}_i) \in \ell$ do
12:                if $\nexists r.r \in \mathsf{R}_n \land r \in \mathsf{R}_i$ then
13:                    $\varphi = \varphi \lor (\varphi_i \land \alpha)$
14:                    if SATCONF($\varphi_i \land \alpha$, $\mathsf{state}_0$) then
15:                        ▷ A violation found
16:                    end if
17:                    $\ell_n = \ell_n \mathbin{++} [(\mathsf{ev}_i, \varphi_i, \mathsf{R}_i)]$
18:                end if
19:            end for
20:            $\mathcal{L}' = \mathcal{L}' \mathbin{++} \ell_n$
21:        end for
22:        return $(\varphi, \mathcal{L}', R_o \setminus R_n)$
23:    end if
24: end function
```

Removing Rules. Incremental verification of rules for a property violation when a rule is removed is handled differently. The algorithm, REMOVEEU, proceeds by checking if pre-state formulas computed earlier violate the property for those rules that have not been deleted from the set of rules R_o. If a violation is detected with the newer set of rules, i.e., $\mathsf{R}_o \setminus \mathsf{R}_n$, it is reported along with the set of rules in the R_n that the user deleted. As the fixed-point formula is a disjunction of the pre-state formulas from the second column in the table, REMOVEEU works by simply deleting the row from the table and updating the final fixed-point formula.

In the example above, suppose the user identifies that the rule R2 should not be active (and deactivates it), and wants to confirm that the desired behavior is not violated any more. Instead of re-verifying all rules, SAFETAP$^{\Delta}$ uses the REMOVEEU functionality to remove R2 from the set of rules, and uses the state-table to verify the behavior. The REMOVEEU function removes the second row in A. corresponding to R2, all three rows in B. and both the rows in C.. It then checks only the rows R1, R3 and R4 in A., which verify successfully satisfying the behavior. The updated state-table is shown below:

A.	False	R1
	Location $\neq$ Home $\land$ Homemode $=$ On	R3
	OccButton $=$ On $\land$ Location $\neq$ Home $\land$ Heater $=$ On	R4

Property type	Example property	Violating rules
Always/Never (Property 1)	Temperature of the house is never greater than 75°	Raise temperature of the home when heater is turned on
Whenever (Property 2)	Whenever my garage door is opened, make sure I receive a notification	Mute phone at work; Mute your phone when battery is low
Only When (Property 3)	Blink lights only if there is smoke in the house	Blink lights when there is an in-game update from your favorite team

TABLE I: Example properties checked for in SafeTAP

VII. Evaluation

We evaluate the performance of the basic analysis tool SafeTAP and the performance gain of our incremental analysis in SafeTAP$^\Delta$.

A. Methodology and system setup

We explain how we collect and process the data and then how SafeTAP is implemented.

Data collection and dataset generation. To collect IFTTT rules, we crawled IFTTT [29] for available services, and then gathered the rules associated with the services by visiting each service page[5]. For each rule, we collected all available information—description, title, trigger title, trigger service, action title, action service, number of downloads, and URL for the rule—and used this dataset in our evaluation. To evaluate SafeTAP's performance, we sampled the 5,000 most downloaded rules from the crawled dataset. Among those, we identified 502 unique actions and triggers. As certain rules are dependent on the arguments passed to the triggers and/or actions, we manually encode the types and reasonable ranges for these arguments (e.g., temperature: 32°F–80°F, security system modes: home and away) along with getters and setters to read and modify the state variables. We also encode the trigger and action events.

We implemented an encoder to generate rule representations that SafeTAP understands. The encoder is also used as part of a simulator for evaluation purposes. The simulator randomly selects a specific number of rules from the dataset to emulate rule installation. Our simulator also identifies chains of rules in the dataset. Two rules form a chain if the action of the first rule triggers the second. Our simulator uses the state variables in rules to construct chains.

Experiment setup. We perform two sets of experiments on a MacBook Pro with 8 GB RAM and 3.1 GHz Intel Core i5 processor, running macOS Catalina 10.15.3. We report the performance numbers over 100 trials. SafeTAP and SafeTAP$^\Delta$ are implemented in Python 3.7.3 over 960 LOC.

a) Performance and scalability of SafeTAP*.:* Our simulator randomly selects rules from the 5,000 rule dataset on which we evaluate how well SafeTAP performs at detecting property violations as we vary the number of rules. We start with 100 rules in the first set and increment the number of rules by 50 for each of the following rounds, for up to 500 rules. We check the properties discussed in Section IV-A (example properties are shown in Table I), and report the time taken to detect the first violation and all possible violations (may be 0). For measuring the time taken to verify the first violations, we manually add/remove rules to induce a violation if no violations of the property are found in the rule-set.

[5]Scraped 259,523 IFTTT rules in October, 2017

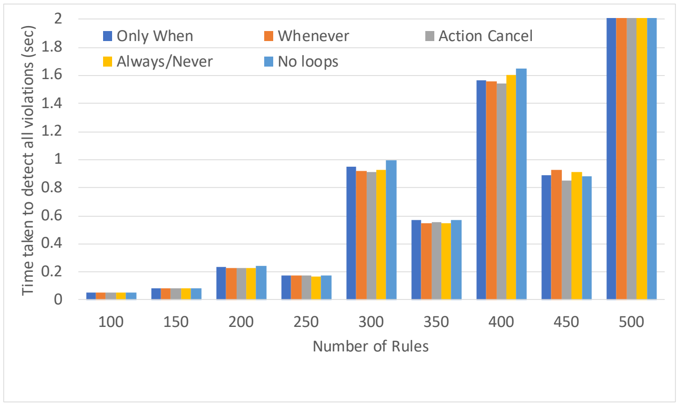


Fig. 4: Performance of SafeTAP: x-axis plots the number of rules in the data-set while y-axis plots the execution time in seconds to detect all violations for different properties.

b) Effect of incremental analysis on the performance.: To investigate the effect of incremental analysis on the performance of SafeTAP$^\Delta$, we perform an experiment varying the number of rules and checking for Only when property in three rounds. In the first round, we perform the initial analysis needed to generate the state. In the second round, we perform the incremental analysis with a couple of new rules added to each set. In the third round, we perform the analysis without using the incremental analysis on the newer rules added to the initial set of rules as would be the case in a normal analysis.

Additionally, we perform an experiment to compare the time taken with incremental analysis for addition and removal of rules in SafeTAP$^\Delta$ with the time taken in original analysis in SafeTAP. We vary the number of rules used in each round. This experiment is performed thrice, once each for 2, 4 and 8 rules being added to and removed from the original rule set.

B. Results of Evaluating SafeTAP and SafeTAP$^\Delta$

The results of our experiments show that SafeTAP is efficient and scales well with a larger number of rules. Interestingly, we observe that the time taken to detect property violations increases with the maximum length of the chains. This is because the length of the paths to be explored increases with the length of the chains, which means SafeTAP takes a longer time to detect a violation with longer chains. Additonally, the incremental algorithm provides a better performance as compared to normal analysis after the initial analysis.

Performance and scalability of SafeTAP. The results of our experiment can be found in Figure 4. The detailed numbers are found in Table VII in the Appendix. For the properties listed in Table I, we include the time taken to check the property until the first violation is found (FV) and the time taken to check the property for all violations (AV) for each dataset.

The time taken to detect the first violation is less than 20

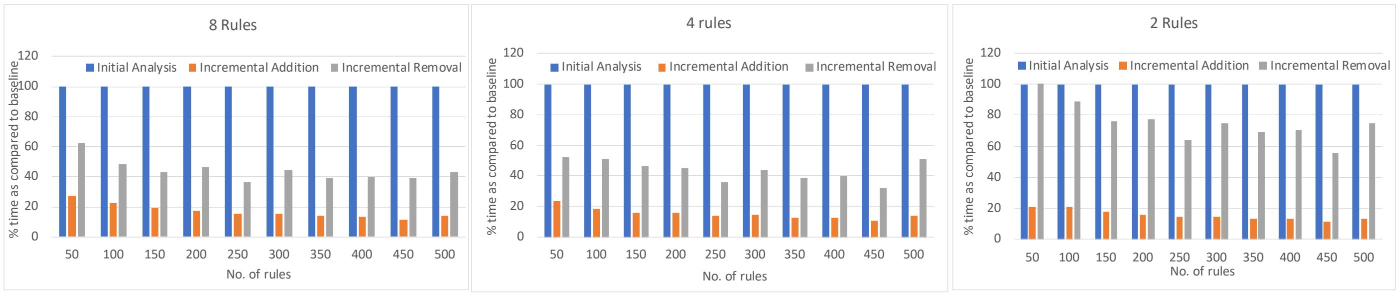


Fig. 5: Performance of SafeTAP$^{\triangle}$ for both incremental addition and removal of rules varying both the number of rules in the original set and the set being added or removed : x-axis plots the number of rules in the data-set while y-axis plots the percentage time taken for initial analysis, incremental addition of rules, and incremental algorithm of rules with initial analysis as the baseline. The first graph shows the results when 8 rules were added and removed, the second graph shows for 4 rules and the third graph shows for 2 rules.

ms for all properties for 100 rules data-set, while the time taken to detect all violations for all properties is less than 60 ms for 100 rules data-set. Note that, the decrease in the execution time, going from 300 to 350 rules and 400 to 450 rules, is due to a reduction in the number of paths explored by SafeTAP. For 300 rules, the number of paths explored by the algorithm is 335 while the same number for 350 rules is 71. Similarly, the number of paths explored for the 400 rule data-set is higher than for the 450 rule data-set. The number of paths explored increases manifold for the 500 rule data-set; hence, the significant slowdown (around 9 sec for all properties).

Performance of SafeTAP$^{\triangle}$. Figure 18 in the Appendix shows the performance evaluation results for the initial analysis and the analyses with and without the incremental algorithm. As the incremental algorithm has to traverse fewer paths and re-compute fewer state formulas, it is more efficient and provides a better performance than the analysis that does not employ the incremental algorithm. Concretely, the analysis without the incremental algorithm takes on an average 6 times more time than the one with the incremental algorithm. However, there is an added cost of storing state shown in Table VIII in the Appendix, which can be seen by comparing the time taken by the initial analysis against the one shown in Table VII; the average overhead for storing state is about 33.75% but the overhead balances out once the incremental analysis kicks in.

Figure 5 shows the time to verify various set of rules, varying by the number, with different rules being added and removed in each iteration. The evaluation shows that incremental analysis with addition of rules is the fastest when adding 2 rules and increases for 4 and 8 rules. For removal of rules, the trend is reversed with removal of 8 rules requiring less than half the time of the analysis without the incremental algorithm (around $55-60\%$ performance increase), and increases to three-fourths of the baseline time for the analysis with 2 rules removed. The not-so-signficant increase in performace when only 2 rules are removed from the rule-set is due to two factors: (1) SafeTAP$^{\triangle}$ incurs an additional cost for storage and retrieval of prior states and (2) if the rules being removed do not interact much with the other rules, the number of pre-state formulas dependent on those rules are also fewer; as the majority of pre-state formulas are not removed, the analysis takes longer time. As an optimization for EU, if the previous property was satisfied, we can directly remove the rules without having to re-check the satisfiability of the remaining rules.

VIII. Discussion

Other properties. Prior work [48] has listed a few other properties that might cause problems in home automation platforms. While some of these properties can be represented as a safety property, others work at a high-level without needing a specific user-defined behavior. For instance, some rules naturally form pairs based on the behavior of their actions. “Start recording” and “stop recording”, and “turn light on” and “turn light off” are two examples of actions which form pairs. When an action is missing from a pair, it can lead to unbounded behavior because the action is never “undone”. For example, the rule, “IF I leave home THEN start recording”, will begin recording when the user leaves home. If there is no matching rule to stop recording, the recording will continue indefinitely. Eventually, the device’s memory will fill and stop recording, whether or not the user is home. SafeTAP$^{\triangle}$ has heuristics for finding un-matched rules and works by checking a counterpart for each action. The CTL property is shown in Appendix B. However, most users might not be interested in such properties as they might have deliberately not installed rules that perform the counter-action.

A rule cannot behave as intended if the action it triggers is immediately reverted. No Undo is similar to the “action revert” property from [48] and the detection of “self disabling” rules by [15]. The smoke detecting rule, “IF smoke detected THEN blink light” would be rendered useless if another rule “IF it is daytime THEN turn off light” immediately turned off the blinking light. A property to ensure that blinking lights are not undone would be useful, here. While SafeTAP$^{\triangle}$ supports checking this property if specified as a CTL property, it does not support a general version that checks for all possible cases.

Triggering the same action several times in quick succession could be annoying, or even dangerous, depending on how sensitive the action is. A similar property has been referred to as “action duplication” [48] and also appear in other work [12], [38]. SafeTAP$^{\triangle}$ does not support such properties yet because the property specification does not account for time as a parameter; instead it only deals with states. Adding support for this is an interesting direction for future work.

IX. Related Work

Formal analysis of TAP. Model checking is a popular technique for verifying properties about TAP for IoT devices and IoT applications. Many prior approaches use general-purpose model checkers [12], [28], [32], [33], [38], [48], [52]; some, like our work, develop and implement their own algorithms [55]. What sets our work apart from the above mentioned model checking tools is SAFETAP$^{\Delta}$, our novel incremental algorithm. This not only allows properties to be re-checked efficiently as users fiddle with their rules, it also provides users with timely and precise feedback that is specific to the TAP rules that caused the change in the analysis results. In particular, SAFETAP$^{\Delta}$'s ability to only report issues caused by rule changes in the presence of existing violations would be hard to achieve using an off-the-shelf model checker; it would require nontrivial modifications which were more straightforward to make in our custom tool.

SAFETAP can analyze many of the properties stated in recent projects AutoTap [55] and Soteria [12]. Currently, SAFETAP does not have support for the temporal properties which AutoTap can verify, but we plan to address this in future work. Soteria can spot apps which generate events without first declaring that they subscribe to them. Our work focuses on TAP rules instead of apps. We do not explicitly support the "inconsistent events" property described in [12], but it is similar to our Only When property.

On the other hand, SAFETAP additionally supports loop detection (which neither AutoTap nor Soteria can do), and action cancelling (which AutoTap can't check). AutoTap focuses on user-defined *safety* (Always, Never, Whenever, Only When) properties and cannot detect loops or action cancelling, while Soteria does not support loop detection, nor do they give special attention to Only When properties. SafeChain [28] encodes privacy leaks and privilege escalation as reachability properties, which is fundamentally different from SAFETAP, AutoTap, and Soteria. Encoding and analyzing such properties is an interesting future direction. We use a fragment of CTL for property specification for efficiency, while AutoTap, Soteria, and SafeChain use LTL for specifying properties.

The main objective of AutoTap is to synthesize rules for users to satisfy desired properties, even though the synthesis algorithm is based on model checking and can be used to verify existing rules. Incorporating their synthesis algorithm to our incremental symbolic model checking algorithm would be an exciting future direction to explore. Soteria [12], meanwhile, evaluates safety and security properties of IoT applications (not TAP rules) by translating their source code to extract a state model for model checking.

Similar to prior work [20], [28], [55], SAFETAP uses a simple model of the physical environment, which consists of only variables representing the state (e.g., temperature). Bu et al. [10] use hybrid automata to accurately model the continuous behavior of environment variables (like temperature). Such a detailed model for physical environment is unnecessarily complicated for identifying high-level problems in TAP rules.

Finally, SAFETAP considers the violations induced by the user's interactions with the devices or changing the environment, while others do not [28], [55].

Mitigating security and safety problems of TAP. In addition to analysis tools for TAP, researchers have also built systems using runtime monitoring to protect the users against potential attacks or to prevent them from reaching an unsafe state or by suggesting fixes to remove problematic rules. For example, Bastys *et al.* proposed a runtime information flow tracking monitor to prevent potential user privacy violations [8]. IoTGuard instruments application code and monitors the app and TAP platform behavior at runtime to ensure they satisfy given security policies [13]. When violations are found, users can ask IoTGuard to automatically block the violating action, or allow or deny the action through a runtime prompt. We proactively analyze safety properties ahead of time. Mitigation is beyond the scope of this work and we leave it for future work.

Smarthome security. Recent research has discovered potential vulnerabilities in IoT devices and smarthome applications [22], [25], [26], [42], [43], [51]. Celik et al. [11] identified sensitive data flows in SmartThings applications using static analysis. SmartAuth [47] identified issues with overprivileged SmartThings applications based on program analysis of the applications and Natural Language Processing (NLP) analysis of the application descriptions. Whereas, Tkperm [41] proposed a transfer learning-based scheme to detect overprivileged applications across different IoT platforms. IoTMon also combines static analysis and NLP analysis to identify chains in SmartThings applications and the risks posed by the chains. Most of the previous papers design security solutions without knowledge of real home usage. To solve this problem, Manandhar et al. propose a framework that generates natural home automation scenarios [34]. This line of work is orthogonal to ours and covers a complementary area.

User studies of smart homes. To understand users' privacy concerns about smart-home technology, several studies investigated users' experiences [3], [5], [7], [14], [19], [21], [46] and preferences with different settings (such as multi-user [23], [49], [54], and guest-host environment [7]). Studies have shown that users have trouble understanding feature interactions of IoT devices [9], [50]. Researchers have created and tested tools using formal methods with end-users in mind for use in other domains [2], [37]. While others [24], [40] have studied desirable access controls for smart-home devices, we perform a user study to understand the effectiveness of SAFETAP in identifying problems in TAP rules.

X. CONCLUSION

This paper investigates the problem of helping users identify undesired behaviors for their trigger-action programs (TAP) used for home automation. We carried out a user study, which shows that users struggle with uncovering issues in their TAP rules and would like to check their rules when they modify their rules. To help give users prompt feedback, we designed and implemented SAFETAP$^{\Delta}$, a novel incremental symbolic model checker for TAP rules. Our evaluation shows that incremental checking on average improves the performance by 6X when adding new rules. Our case study highlights the difficulty that users have with identifying problems with TAP rules, especially as the size of the rule sets increase. Moreover, we found that people overestimated their abilities to understand TAP rule interactions, and were reluctant to use a tool to help them make decisions about whether rules behave as expected. This highlights the need for research not only about how to build tools to help users reason about TAP rule interactions, but how to make such tools *desirable* to the users so that they are actually used.

References

[1] "Amazon Mechanical Turk," https://www.mturk.com/, June 06 2020.

[2] "Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/soups2020/presentation/smith

[3] N. Abdi, K. M. Ramokapane, and J. M. Such, "More than smart speakers: Security and privacy perceptions of smart home personal assistants," in *Proceedings of the 15th Symposium on Usable Privacy and Security (SOUPS)*, 2019.

[4] Apple Inc., "iOS - Home - Apple," 2020. [Online]. Available: https://www.apple.com/ios/home/

[5] N. Apthorpe, Y. Shvartzshnaider, A. Mathur, D. Reisman, and N. Feamster, "Discovering smart home internet of things privacy norms using contextual integrity," *Proceedings of ACM Interaction Mobile Wearable Ubiquitous Technology*, vol. 2, no. 2, pp. 59:1–59:23, 2018.

[6] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: The MIT Press, 2008.

[7] N. M. Barbosa, Z. Zhang, and Y. Wang, "Do privacy and security matter to everyone? quantifying and clustering user-centric considerations about smart home device adoption."

[8] I. Bastys, M. Balliu, and A. Sabelfeld, "If This Then What? Controlling Flows in IoT Apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1102–1119. [Online]. Available: https://doi.org/10.1145/3243734.3243841

[9] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, "How users interpret bugs in trigger-action programming," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3290605.3300782

[10] L. Bu, W. Xiong, C.-J. M. Liang, S. Han, D. Zhang, S. Lin, and X. Li, "Systematically ensuring the confidence of real-time home automation iot systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 2, no. 3, Jun. 2018. [Online]. Available: https://doi.org/10.1145/3185501

[11] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive information tracking in commodity iot," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. USA: USENIX Association, 2018, pp. 1687–1704.

[12] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '18. USA: USENIX Association, 2018, pp. 147–158.

[13] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proceedings of the 23rd Network and Distributed Security Symposium*. USA: Internet Society, 2019.

[14] G. Chalhoub, I. Flechais, N. Nthala, and R. Abu-Salma, "Innovation inaction or in action? the role of user experience in the security and privacy design of smart home cameras," in *Proceedings of the Symposium on Usable Privacy and Security*. ACM Digital Library, 2020.

[15] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," *CoRR*, vol. abs/1808.02125, 2018. [Online]. Available: http://arxiv.org/abs/1808.02125

[16] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: A new symbolic model verifier," in *International conference on computer aided verification*. Springer, 1999, pp. 495–499.

[17] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: The MIT Press, 1999.

[18] C. Cobb, M. Surbatovich, A. Kawakami, M. Sharif, L. Bauer, A. Das, and L. Jia, "How risky are real users ifttt applets?" in *Proceedings of the 16th Symposium on Usable Privacy and Security (SOUPS)*, 2020.

[19] ——, "How risky are real users ifttt applets?" 2020.

[20] R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl, "Model checking information flow in reactive systems," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2012, pp. 169–185.

[21] P. Emami-Naeini, H. Dixon, Y. Agarwal, and L. F. Cranor, "Exploring how privacy and security factor into IoT device purchase behavior," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI)*, 2019.

[22] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, pp. 531–548.

[23] C. Geeng and F. Roesner, "Who's in control?: Interactions in multi-user smart homes," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI)*, 2019, pp. 268:1–268:13.

[24] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home internet of things (IoT)," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, 2018, pp. 255–272.

[25] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter, "A policy language for distributed usage control," in *Computer Security – ESORICS 2007*, J. Biskup and J. López, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 531–546.

[26] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 461–472. [Online]. Available: https://doi.org/10.1145/2897845.2897886

[27] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2004, vol. 1003.

[28] K.-H. Hsu, Y.-H. Chiang, and H.-C. Hsiao, "Safechain: Securing trigger-action programming from attack chains," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2607–2622, 2019.

[29] IFTTT, "IFTTT: Every thing works better together," 2020. [Online]. Available: https://ifttt.com

[30] James A. Martin and Matthew Finnegan, "What is IFTTT? How to use If This, Then That services," 2019. [Online]. Available: https://www.computerworld.com/article/3239304/what-is-ifttt-how-to-use-if-this-then-that-services.html

[31] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International journal on software tools for technology transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[32] C.-J. M. Liang, L. Bu, Z. Li, J. Zhang, S. Han, B. F. Karlsson, D. Zhang, and F. Zhao, "Systematically debugging iot control system correctness for building automation," in *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, ser. BuildSys '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 133–142. [Online]. Available: https://doi.org/10.1145/2993422.2993426

[33] C.-J. M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu, "Sift: Building an internet of safe things," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, ser. IPSN '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 298–309. [Online]. Available: https://doi.org/10.1145/2737095.2737115

[34] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Towards a natural perspective of smart homes for practical security and safety analyses," in *Proc. of 41st IEEE Symposium on Security and Privacy, San Francisco, CA, USA*, 2020, pp. 1–18.

[35] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[36] Microsoft, "Microsoft Power Automate," 2020. [Online]. Available: https://flow.microsoft.com

[37] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, pp. 97–102.

[38] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M.

Colbert, and P. McDaniel, "IotSan: Fortifying the Safety of IoT Systems," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 191–203. [Online]. Available: https://doi.org/10.1145/3281411.3281440

[39] openHAB Foundation e.V., "openHAB," 2020. [Online]. Available: https://www.openhab.org/

[40] R. Schuster, V. Shmatikov, and E. Tromer, "Situational access control in the Internet of Things," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 1056–1073.

[41] F. Shezan, K. Cheng, Z. Zhang, Y. Cao, and Y. Tian, "Tkperm: Cross-platform permission knowledge transfer to detect overprivileged third-party applications," in *Proceedings of the 27th Network and Distributed Security Symposium*. Internet Society, 01 2020.

[42] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-level security and privacy control for smart-home iot devices," in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. USA: IEEE Press, Oct 2015, pp. 163–167.

[43] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, "Smart-phones attacking smart-homes," in *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 195–200. [Online]. Available: https://doi.org/10.1145/2939918.2939925

[44] SmartThings Inc., "SmartThings," 2019. [Online]. Available: https://www.smartthings.com

[45] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, pp. 1501–1510. [Online]. Available: https://doi.org/10.1145/3038912.3052709

[46] M. Tabassum, T. Kosinski, and H. R. Lipford, ""I don't own the data": End user perceptions of smart home device data practices and risks," in *Proceedings of the 15th Symposium on Usable Privacy and Security (SOUPS)*, 2019.

[47] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. USA: USENIX Association, 2017, pp. 361–378.

[48] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1439–1453. [Online]. Available: https://doi.org/10.1145/3319535.3345662

[49] Y. Yao, J. R. Basdeo, O. R. Mcdonough, and Y. Wang, "Privacy perceptions and designs of bystanders in smart homes," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 59:1–59:24, Nov. 2019.

[50] S. Yarosh and P. Zave, "Locked or not? mental models of iot feature interaction," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 2993–2997. [Online]. Available: https://doi.org/10.1145/3025453.3025617

[51] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2834050.2834095

[52] B. Yuan, Y. Jia, L. Xing, D. Zhao, X. Wang, D. Zou, H. Jin, and Y. Zhang, "Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/yuan

[53] Zapier Inc., "Zapier - The easiest way to automate your work," 2020. [Online]. Available: https://zapier.com/

[54] E. Zeng and F. Roesner, "Understanding and improving security and privacy in multi-user smart homes: A design exploration and in-home user study," in *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*, 2019, pp. 159–176.

[55] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "Autotap: Synthesizing and repairing trigger-action programs using ltl properties," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. USA: IEEE Press, 2019, pp. 281–291. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00043

Appendix

A. Modeling TAP

Event	ev	$::=$	New email received | Presence detected | ...
Expressions	e	$::=$	$c \mid s \mid e_1$ bop e_2
Predicates	p	$::=$	e_1 cop $e_2 \mid$ ev
Literals	l	$::=$	$\top \mid \bot \mid p \mid \neg p$
State formulas	α, β	$::=$	$\bigwedge_{i=i}^{n} l_i$
State	σ	$::=$	$\cdot \mid \sigma, s \mapsto c$
Rule	r	$::=$	$(\text{ev}, \alpha) \Rightarrow (\text{ev}', U, \mathcal{E}_{ff})$
Rules	R	$::=$	$\cdot \mid R, r$
Updates	U	$::=$	$\cdot \mid U, s := e$
Physical effects	$\mathcal{E}_{ff}$	$::=$	$\cdot \mid \mathcal{E}_{ff}, s \uparrow \mid \mathcal{E}_{ff}, s \downarrow \cdots$
Event queue	E	$::=$	$\cdot \mid E$, ev
Configuration	C	$::=$	$(\sigma, E, \mathcal{E}_{ff})$

Fig. 6: Syntax for modeling TAP

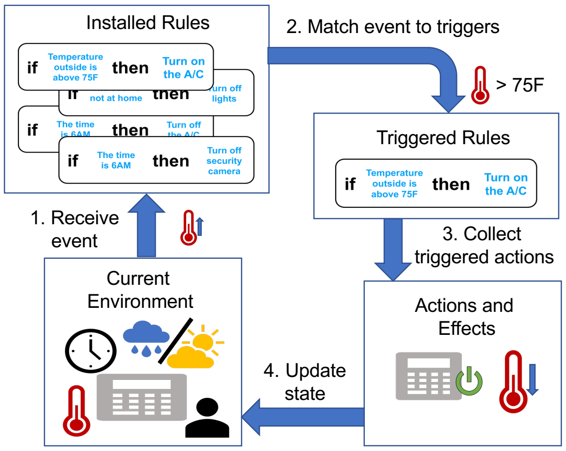

Installed Rules
if Temperature outside is above 75F then Turn on the A/C
if not at home then Turn off lights
if The time is 6AM then Turn off the A/C
if The time is 6AM then Turn off security camera
2. Match event to triggers
> 75F
Triggered Rules
if Temperature outside is above 75F then Turn on the A/C
1. Receive event
3. Collect triggered actions
Current Environment
Actions and Effects
4. Update state

Fig. 7: Informal Semantics for Modeling TAP

We model the trigger-action platform as a labeled transition system [6], which consists of states and transitions. The syntax for modeling the trigger-action platform is shown in Figure 6. Expressions e include environment or device state variables s (e.g., temp, lightColor, sysArmed), constants c (e.g., 70, "blue", true), and binary operations on two expressions. An event ev indicates a change in state—"New email received", "Heater on and user home", and "Phone not charging" are all valid events. The events themselves do not carry any parameters, e.g., Nest temperature is $80°$. Instead the state formula α capture these parameters using state variables.

Predicates are facts about the system that are expressed using comparisons on expressions (e_1 cop e_2). A state formula, α, is a fact that can be determined by looking at a single state; the predicates temp $>$ 70 and doorLocked $=$ true are both valid state formulas.

We write σ to denote the current state of the physical environment and IoT devices. σ maps state variables, s, to concrete values, c. A valid state could be $\sigma = [\mathit{NestTemp} \mapsto 70; \mathit{systemArmed} \mapsto \text{true}]$, indicating that currently, the Nest temperature reading is $70°$ and the system is currently armed.

A trigger-action rule is denoted $(\text{ev}, \alpha) \Rightarrow (\text{ev}', U, \mathcal{E}_{ff})$, where (ev, α) to the left of the double arrow represents the triggering event and the tuple $(\text{ev}', U, \mathcal{E}_{ff})$ to the right of the double arrow represents the effect of the action. Here ev is the triggering event and α describes additional conditions that must hold in the current state for the rule to be triggered. For example, a user turning on the heater can be represented as $(\text{USER}, \mathit{temp} < 68)$. While USER is the triggering event corresponding to the user triggering this rule, $\mathit{temp} < 68$ is an additional condition which checks if the current temperature is less than $68°$ to trigger the rule; if not, the rule is not triggered.

The full effect of the action is represented as $(\text{ev}', U, \mathcal{E}_{ff})$. Here ev$'$ is the new event, if any, that is produced by the action, U is the set of updates to state variables caused by the action, and $\mathcal{E}_{ff}$ is the set of resulting environmental effects. For example, the action of turning on the heater will be encoded as $(\text{HEATERON}, \mathit{heater} := \text{On}, \mathit{temp} \uparrow)$; HEATERON is the new event while the state of heater is set to On and the temperature is set to rising. Note that the effect of turning on the heater on the temperature does not include a new temperature as a concrete value, so the effect is abstractly encoded as temperature rising ($\mathit{temp} \uparrow$).

To model the effect of $\mathcal{E}_{ff}$ on the state variables, we include a function that modifies the state depending on what effect is encoded in $\mathcal{E}_{ff}$. For instance, to model the effect of $\mathit{temp} \uparrow$ on temp, we require a function that raises the temperature by $n°$, where n is the change in temperature that occurs as the system transitions to the next state. Our model over-approximates and raises the value of the state variable to the maximum possible value given by the range of the state variable, for optimization purposes. In this example, instead of increasing the temperature by $n°$ for every state transition, our algorithm raises it to the highest possible temperature, given by a pre-determined range in the encoding library (e.g., $100°$). This is sufficient to model changes to the physical environment without sacrificing efficiency by updating it one state at a time.

The system configuration, C includes a snapshot of the current environmental state, σ, a queue of events to be processed, E, along with list of current physical effects, $\mathcal{E}_{ff}$. The queue of events is to keep track of any additional events generated from triggered rules because this is how devices interact with each other in the trigger-action platform.

The operational semantics for rule evaluation use the judgement: $R \vdash C \longrightarrow C'$ as shown in rule ProcEvent in Figure 8. C' is the resulting configuration when rules R receive the event ev under configuration C. The notation (ev, E) and (E, ev) indicate a list of events with ev as first element and last element, respectively, while the notation $E@E'$ indicates the concatenation of two lists E and E'. The triggering of each rule is evaluated by the judgement $R, \sigma, \text{ev}, \mathcal{E}_{ff} \Downarrow \sigma', E', \mathcal{E}'_{ff}$, which is defined inductively.

Next, we explain the how TAP rules are triggered in the semantics. Rule R-Trig triggers a TAP rule r, if ev matches the event in the TAP rule trigger and the state formula α holds in the current state. The action of the triggered TAP rule updates the physical effects in the environment along with the state variables, and may trigger another event. The function updEnv updates the environmental effects to $\mathcal{E}'_{ff}$ with the effects generated by the rule, $\mathcal{E}^{r}_{ff}$, while the function updState updates the state variables with the updates from the rule, U, and the new environmental effects, $\mathcal{E}'_{ff}$. This generates

$$\text{R-EMP}\ \frac{}{[], \sigma, \mathsf{ev}, \mathcal{E}_{\mathit{ff}} \Downarrow \sigma, [], \mathcal{E}_{\mathit{ff}}}$$

$$\text{R-TRIG}\ \frac{\begin{array}{c} r = (\mathsf{ev}, \alpha) \Rightarrow (\mathsf{ev}', U, \mathcal{E}^{r}_{\mathit{ff}}) \\ \mathsf{isTrue}(\sigma, \alpha) \qquad \mathcal{E}'_{\mathit{ff}} = \mathsf{updEnv}(\mathcal{E}_{\mathit{ff}}, \mathcal{E}^{r}_{\mathit{ff}}) \\ (\sigma', E') = \mathsf{updState}(\sigma, U, \mathcal{E}'_{\mathit{ff}}) \\ R, \sigma', \mathsf{ev}, \mathcal{E}'_{\mathit{ff}} \Downarrow \sigma'', E'', \mathcal{E}''_{\mathit{ff}} \end{array}}{(r, R), \sigma, \mathsf{ev}, \mathcal{E}_{\mathit{ff}} \Downarrow \sigma'', (\mathsf{ev}, E'@E''), \mathcal{E}''_{\mathit{ff}}}$$

$$\text{R-SKIP}\ \frac{\begin{array}{c} r = (\mathsf{ev}', \alpha) \Rightarrow \_ \qquad (\mathsf{ev}' \neq \mathsf{ev} \vee \neg \mathsf{isTrue}(\sigma, \alpha)) \\ R, \sigma, \mathsf{ev}, \mathcal{E}_{\mathit{ff}} \Downarrow \sigma', E', \mathcal{E}_{\mathit{ff}} \end{array}}{(r, R), \sigma, \mathsf{ev}, \mathcal{E}_{\mathit{ff}} \Downarrow \sigma', E', \mathcal{E}_{\mathit{ff}}}$$

$$\text{PROCEVENT}\ \frac{R, \sigma, \mathsf{ev}, \mathcal{E}_{\mathit{ff}} \Downarrow \sigma', E', \mathcal{E}'_{\mathit{ff}}}{R \vdash (\sigma, (\mathsf{ev}, E), \mathcal{E}_{\mathit{ff}}) \longrightarrow (\sigma'', E@E', \mathcal{E}'_{\mathit{ff}})}$$

Fig. 8: Semantics of rule evaluation.

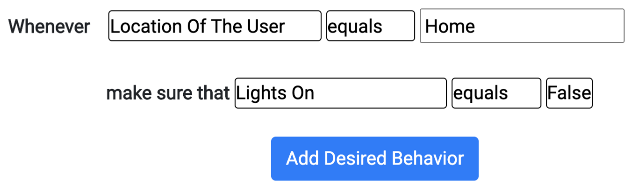
an updated state σ' and might trigger new events E'. The evaluation of R-TRIG results in a modified state σ'', a new event-queue $E'@E''$ and modified physical effects $\mathcal{E}'_{\mathit{ff}}$. Rule R-SKIP handles the case when the triggering event does not match the TAP rule's trigger or the pre-condition is not true, in which case it proceeds with checking the rest of the TAP rules R. Rule R-EMP states that when the list of TAP rules is empty (i.e. all rules have been checked), the event has been completely processed.

B. SAFETAP *Properties*

The various properties that can be analyzed by SAFETAP are shown in the table in Figure 9.

C. *Web Interface*

A Whenever property which makes sure that the lights are off whenever the user is away can be entered in the following template:



Whenever | Location Of The User | equals | Home

make sure that | Lights On | equals | False

D. *Algorithms*

1) Algorithm to check EU *properties in* SAFETAP$^{\Delta}$*:* The top-level function for checking EU properties, CHECKEU, is shown in Algorithm 4. It takes as input state formulas α and β, a set of rules, and the initial configuration against which the formula $\mathsf{E}(\alpha\ \mathsf{U}\ \beta)$ needs to be checked. The algorithm works by repeatedly computing the pre-condition for the given property and the set of rules until a fixpoint is reached. It does so by calling INCADDEU

Algorithm 4 Algorithm to check $\mathsf{E}(\alpha\ \mathsf{U}\ \beta)$ formulas

```
function CHECKEU(state_0, $\alpha$, $\beta$, R)
    INCADDEU(state_0, $\alpha$, $\beta$, (False, [], []), R)
end function
```

Algorithm 5 Algorithm to check $\mathsf{EF}(\beta)$ formulas

```
function CHECKEF(state_0, $\beta$, R)
    CHECKEU(state_0, True, $\beta$, R)
end function
```

2) Algorithm to check EG *formula:* The function CHECKEG tests the satisfiability of the initial configuration state_0 in the set of states satisfying *all* the pre-state formulas returned by SIMPLIFYEG. φ_n is a conjunction over all pre-state formulas corresponding to all possible (pre-)states satisfying φ. If the function SATCONF returns true, a violation is found. The function SIMPLIFYEG (Algorithm 7), similar to SIMPLIFYEU, generates a list of pre-state event and formula pairs given the set of rules R and the initial formula φ and continues to generate pre-state formulas until it reaches a fixpoint, which generates the final pre-state formula.

Algorithm 6 Algorithm to check $\mathsf{EG}(\alpha)$ formulas

```
function CHECKEG(state_0, $\alpha$, ($\varphi$, $\mathcal{L}$, R_old), R_new)
    ($\varphi_n$, $\mathcal{L}'$, R) = SIMPLIFYEG($\alpha$, ($\varphi$, $\mathcal{L}$, R_old), R_new)
    if SATCONF($\varphi_n$, state_0) then
        return True
    end if
    return False
end function
```

Algorithm 10 describes the algorithm for checking $\mathsf{EF}(\alpha \wedge \mathsf{EF}(\beta))$ formulas.

E. *User Study Results*

F. *Evaluation*

Property	Description	CTL formula	SafeTAP formula
Property 1: Always/Never	Always α Never α	$\mathsf{AG}(\alpha)$ $\mathsf{AG}(\neg\alpha)$	$\neg\mathsf{E}(\top\ \mathsf{U}\ \alpha)$ $\neg\mathsf{E}(\top\ \mathsf{U}\ (\neg\alpha))$
Property 2: Whenever	Whenever α, make sure that β	$\mathsf{AG}(\alpha \rightarrow \beta)$	$\mathsf{E}(\top\ \mathsf{U}\ (\alpha \land \neg\beta))$
Property 3: Only When	α only when β	$\mathsf{AG}(\beta \rightarrow \alpha)$	$\neg\mathsf{E}(\top\ \mathsf{U}\ (\beta \rightarrow \alpha))$
Property 4: No Loops	-	-	-
Property 5: No Action Cancelling	-	-	-
No Missing Rules	α has matching condition β	$\mathsf{AG}(\alpha \rightarrow \mathsf{AF}\ \beta)$	$\mathsf{EF}(\alpha \land \mathsf{EG}(\beta))$

Fig. 9: CTL and SafeTAP formulas for each property from Section IV-A

IoT Experience	Own (23), Informed (1)
Number of IoT Device	0 (1), 1-2 (30), 3-5 (7), 6-9 (1), 10+(1)
Years of IoT Use(years)	Less than a year (4), 1-2 (23), 3-5 (12), 6-9 (1)
IoT Interaction Frequency	Daily (8), Most days (21), Once a week (4), Rarely (1)
Home Automation Usage	Yes (22), No (18)
Number of Home Automation Rules	1-10 (17), 10-25 (5)
Home Automation Rule Update Frequency	Every time I add new rules (7), Every time I get a new IoT device (5), Periodically (4), Infrequently, but more than never(5)
Any Unexpected Rules	Yes (5), No (35)
Want Service to Specify Behavior	Yes (20), No (4), Maybe (16)
Service to Specify Behavior Frequency of Use	Every time I removed or updated a rule (10), Every time I installed a new rule (17), Infrequently, but not never (9), I would install rules, but I would never use this service (2), I would never install rules, so I would never use this service(2)
The tasks were easy	Strongly Agree (6), Agree (17), Somewhat Agree (13), Neither Agree Nor Disagree (1),Somewhat Disagree (4), Disagree (0), Strongly Disagree (0)
The tasks would be harder if there were more rules	Strongly Agree(4), Agree (11), Somewhat Agree(12), Neither Agree Nor Disagree(4),Somewhat Disagree(5), Disagree(2), Strongly Disagree(2)
The rule sets in the tasks seemed realistic	Strongly Agree(14), Agree (16), Somewhat Agree(10), Neither Agree Nor Disagree(),Somewhat Disagree(), Disagree(), Strongly Disagree()
The desired behaviors in the tasks seemed realistic	Strongly Agree(9), Agree (14), Somewhat Agree(12), Neither Agree Nor Disagree(1),Somewhat Disagree(3)
Age	18-20 (1), 21-25 (3), 26-30 (11), 31-35 (7), 36-40 (7), 41-45 (6), 46 or older (5)
Gender	Male (25), Female (15)
Education	High school graduate (1), Some college No degree (10), Associates / 2 year degree (6), Bachelor / 4 year degree (18), Graduate degree Master, PhD, professional, medicine, etc (5)
English Fluency	I exclusively speak, read, and write in English. (34), I speak, read, and write in English and another language, about equally. (5), I primarily speak, read, and write in English sometimes, but use a different language other times. (1)

TABLE II: Summary of participant demographics.

statement	strongly agree	agree	somewhat agree	neutral	somewhat disagree	disagree	strongly disagree
It was easy to add Desired Behaviors for the Tasks	7	3	8	0	0	1	0
After running SafeTAP, it was easy to understand the results	7	8	4	3	0	0	0
I need more instruction to use SafeTAP	2	2	2	4	2	7	2
SafeTAP's interface was confusing	1	0	6	2	2	8	3

TABLE III: SafeTAP experience questions.

statement	strongly agree	agree	somewhat agree	neutral	somewhat disagree	disagree	strongly disagree	I don't use home automation
I would use SAFETAP to help make decisions about my rules	0	13	7	1	0	0	0	1
SAFETAP would make me feel more confident about my rules	0	10	9	1	1	0	1	0
Using SAFETAP would take too long	0	1	3	3	0	14	0	1

TABLE IV: SAFETAP experience questions, cont.

statement	Whenever	Only When	Never	Always	No Loops	No Action Cancelling
SAFETAP: Useful Behaviors	11	18	4	9	8	4
SAFETAP: Useless Behaviors	3	4	6	6	3	3

TABLE V: SAFETAP experience questions, cont.

statement	strongly agree	agree	somewhat agree	neutral	somewhat disagree	disagree	strongly disagree
The tasks were easy	6	17	13	1	4	0	0
The tasks would be harder if there were more rules	4	11	12	4	5	2	2
The rule sets in the tasks seemed realistic	14	16	10	0	0	0	0
The desired behaviors in the tasks seemed realistic	9	14	12	1	3	0	0

TABLE VI: Survey experience questions

# Rules	Only when		Whenever		Action Cancelling		Always/Never		No Loops	
	FV	AV	FV	AV	FV	AV	FV	AV	FV	AV
100	0.0195	0.0522	0.0196	0.0547	0.0062	0.0567	0.0055	0.0560	0.0062	0.0548
150	0.0366	0.0818	0.0352	0.0815	0.0085	0.0809	0.0083	0.0817	0.0095	0.0829
200	0.0603	0.2321	0.0569	0.2260	0.0121	0.2269	0.0109	0.2305	0.0124	0.2461
250	0.0745	0.1752	0.0724	0.1714	0.0174	0.1714	0.0159	0.1706	0.0206	0.1725
300	0.1044	0.9462	0.1012	0.9223	0.0229	0.9085	0.0213	0.9234	0.0227	0.9922
350	0.1250	0.5677	0.1267	0.5498	0.0235	0.5532	0.0294	0.5474	0.0242	0.5703
400	0.1699	1.5655	0.1681	1.5601	0.0254	1.5420	0.0256	1.6007	0.0289	1.6513
450	0.0285	0.8858	0.2043	0.9291	0.0288	0.8534	0.0306	0.9152	0.0338	0.8809
500	0.2425	8.9832	0.2534	8.9181	0.0305	8.8936	0.0295	8.9471	0.0331	9.7784

TABLE VII: Performance results of SAFETAP for different properties. All execution times are in seconds. FV - Time taken to detect first violation; AV - Time taken to detect all violations

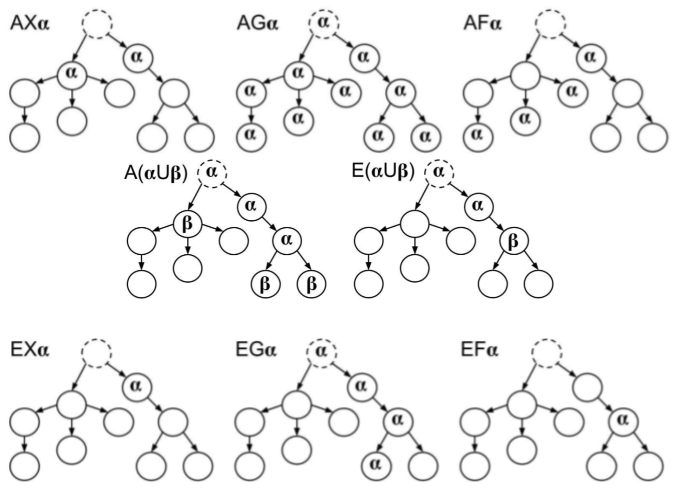
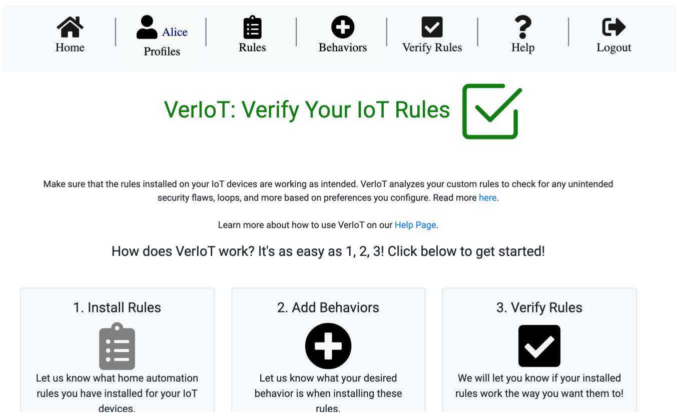


Fig. 10: CTL formula semantics. The current state is indicated by a dotted circle.



VerIoT: Verify Your IoT Rules

Make sure that the rules installed on your IoT devices are working as intended. VerIoT analyzes your custom rules to check for any unintended security flaws, loops, and more based on preferences you configure. Read more here.

Learn more about how to use VerIoT on our Help Page.

How does VerIoT work? It's as easy as 1, 2, 3! Click below to get started!

1. Install Rules

Let us know what home automation rules you have installed for your IoT devices.

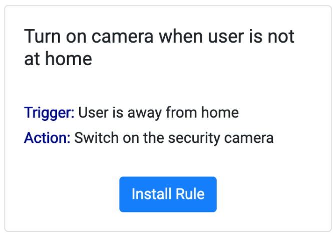
2. Add Behaviors

Let us know what your desired behavior is when installing these rules.

3. Verify Rules

We will let you know if your installed rules work the way you want them to!

Fig. 11: SAFETAP Web Interface Home Page



Turn on camera when user is not at home

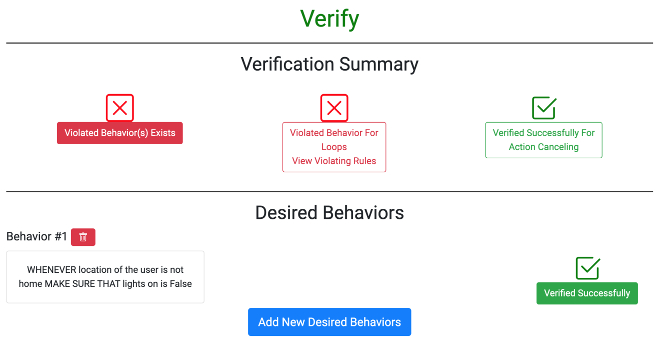
Trigger: User is away from home

Action: Switch on the security camera

Fig. 12: SAFETAP Rule Installation Page

No. of rules	Time taken for initial analysis	Time taken for second analysis with incremental algorithm	Time taken for second analysis without incremental algorithm	% overhead of non-incremental analysis	% overhead for storing state in the first analysis	% gain in second analysis with incremental algorithm
100	0.0665	0.0124	0.0689	455.6451613	27.39463602	24.42528736
150	0.097	0.0169	0.1067	531.3609467	18.58190709	30.37897311
200	0.3268	0.051	0.3331	553.1372549	40.80137872	18.61266695
250	0.1997	0.0277	0.2058	642.9602888	13.98401826	35.10273973
300	1.365	0.1963	1.4023	614.3657667	44.26125555	17.49630099
350	0.7528	0.0951	0.7681	707.6761304	32.60524925	25.32147261
400	2.2026	0.2858	2.2828	698.7403779	40.69626317	20.52379431
450	1.0578	0.1167	1.0881	832.3907455	19.41747573	33.70399639
500	14.914	2.0856	15.046	621.4230917	66.02101701	5.381155936

TABLE VIII: Performance evaluation of incremental analysis for Only when property. All execution times are in seconds. The time with and without incremental analysis is for a second analysis with a couple of extra rules added to each set.


Verify
Verification Summary
Violated Behavior(s) Exists
Violated Behavior For Loops
View Violating Rules
Verified Successfully For Action Canceling
Desired Behaviors
Behavior #1
Verified Successfully
Add New Desired Behaviors

WHENEVER location of the user is not home MAKE SURE THAT lights on is False

Fig. 13: SAFETAP Verification Page

Algorithm 7 Algorithm to simplify $\mathsf{EG}(\alpha)$ formulas

```
function SIMPLIFYEG($\alpha$, $(\varphi, \mathcal{L}, \mathsf{R}_{\mathsf{old}})$, $\mathsf{R}_{\mathsf{new}}$)
    $\varphi_n = \varphi \wedge \alpha$
    $\mathsf{R}_{\mathsf{new}} = \mathsf{R}_{\mathsf{new}} \ (\mathsf{R}_{\mathsf{new}} \cap \mathsf{R}_{\mathsf{old}})$
    $\mathsf{R}_{\mathsf{all}} = \mathsf{R}_{\mathsf{new}} \cup \mathsf{R}_{\mathsf{old}}$
    $\mathcal{L}' = \ell'_n = []$
    $\ell_n$ = PREDSTATELIST($[(\mathsf{noop}, \alpha, [])]$, $\mathsf{R}_{\mathsf{new}}$)
    if $\mathcal{L} \neq []$ then
        for all $\ell \in \mathcal{L}$ do
            $\ell' = \ell \mathbin{++} \ell_n \mathbin{++} \ell'_n$
            $\varphi_n = \varphi_n \wedge (\bigvee_{(\_,\varphi_i,\_)\in\ell'} \varphi_i)$
            $\ell'_n$ = PREDSTATELIST($\ell$, $\mathsf{R}_{\mathsf{new}}$)
            $\ell_n$ = PREDSTATELIST($\ell_n$, $\mathsf{R}_{\mathsf{all}}$)
            $\mathcal{L}' = \mathcal{L}' \mathbin{++} [\ell']$
        end for
        $\ell_n = \ell_n \mathbin{++} \ell'_n$
    end if
    repeat
        $\varphi_o = \varphi_n$
        $\varphi_n = \varphi_o \wedge (\bigvee_{(\_,\varphi_i,\_)\in\ell_n} \varphi_i)$
        $\mathcal{L}' = \mathcal{L}' \mathbin{++} [\ell_n]$
        $\ell_n$ = PREDSTATELIST($\ell_n$, $\mathsf{R}_{\mathsf{all}}$)
    until $\varphi_n \to \varphi_o$ and $\varphi_o \to \varphi_n$
    return $(\varphi_n, \mathcal{L}', \mathsf{R}_{\mathsf{all}}$
end function
```

Algorithm 8 Algorithm to verify $\mathsf{EG}(\alpha)$ when rules are removed

```
function REMOVEEG($\mathsf{state}_0$, $\alpha$, $(\_, \mathcal{L}, \mathsf{R}_o)$, $\mathsf{R}_n$)
    if $\mathcal{L} \neq []$ then
        $\mathcal{L}' = []$
        $\varphi = \mathsf{True}$
        for all $\ell \in \mathcal{L}$ do
            $\ell_n$ = []
            $\varphi_0 = \mathsf{False}$
            for all $(\mathsf{ev}_i, \varphi_i, \mathsf{R}_i) \in \ell$ do
                if $\nexists r. r \in \mathsf{R}_n \wedge r \in \mathsf{R}_i$ then
                    $\varphi_0 = \varphi_0 \vee \varphi_i$
                    $\ell_n = \ell_n \mathbin{++} [(\mathsf{ev}_i, \varphi_i, \mathsf{R}_i)]$
                end if
            end for
            $\varphi = \varphi \wedge \varphi_0$
            $\mathcal{L}' = \mathcal{L}' \mathbin{++} \ell_n$
        end for
        if SATCONF($\varphi$, $\mathsf{state}_0$) then
            ▷ A violation found
        end if
    end if
    return $(\varphi, \mathcal{L}', \mathsf{R}_o \setminus \mathsf{R}_n)$
end function
```

Algorithm 9 Algorithm to check $\mathsf{EF}(\alpha \wedge \mathsf{EG}(\beta))$ formulas

```
function CHECKEFEG($\alpha$, $\beta$, $\mathsf{R}$, $\mathsf{state}_0$)
    $(\beta', \_, \_)$ = SIMPLIFYEG($\beta$, $\mathsf{R}$)
    CHECKEF($\mathsf{state}_0$, $\alpha \wedge \beta'$, $\mathsf{R}$)
end function
```

Algorithm 10 Algorithm to check $\mathsf{EF}(\alpha \wedge \mathsf{EF}(\beta))$ formulas

```
function CHECKEFEF($\alpha$, $\beta$, $\mathsf{R}$, $\mathsf{state}_0$)
    $(\beta', \_, \_)$ = CHECKEU($\mathsf{state}_0$, True, $\beta$, $\mathsf{R}$)
    CHECKEF($\mathsf{state}_0$, $\alpha \wedge \beta'$, $\mathsf{R}$)
end function
```

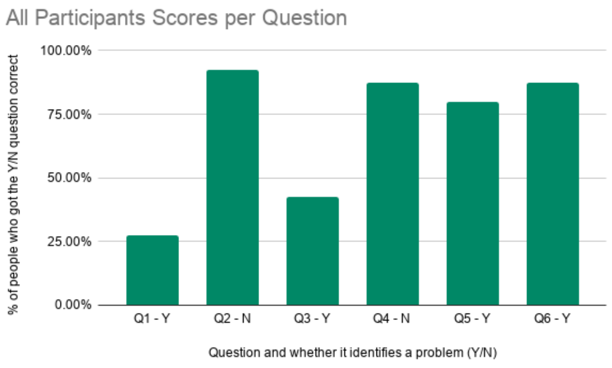


Fig. 14: Average scores for all participants, per question

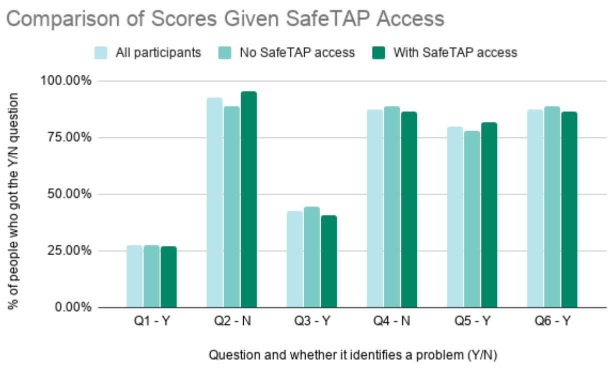


Fig. 17: Average scores for all participants with home automation experience vs without

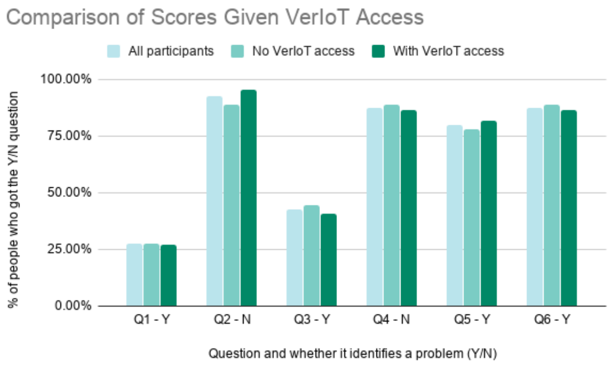


Fig. 15: Average scores for all participants with access to SAFETAP vs without

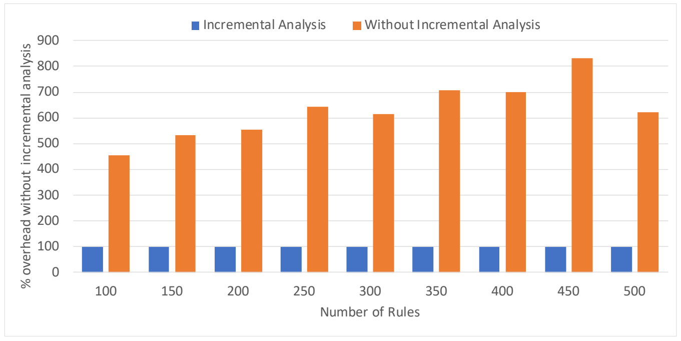


Fig. 18: Performance of SAFETAP$^{\Delta}$: x-axis plots the number of rules in the data-set while y-axis plots the percentage overhead for a subsequent analysis without the incremental algorithm with SAFETAP$^{\Delta}$ as the baseline.

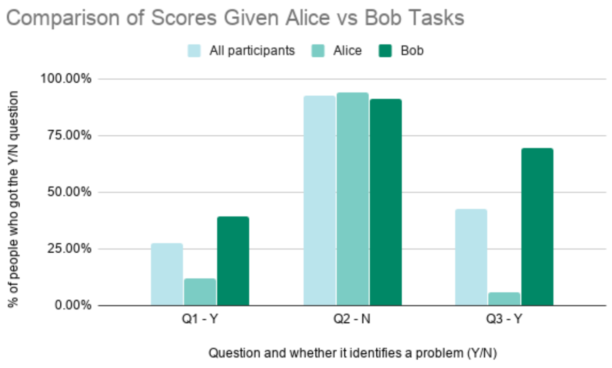


Fig. 16: Average scores for all participants with Alice task vs Bob task