Practical Trigger-Action Programming in the Smart Home

Blase Ur†, Elyse McManus*, Melwyn Pak Yong Ho*, Michael L. Littman*
†Carnegie Mellon University *Brown University
bur@cmu.edu, elyse_mcmanus@brown.edu, melwyn_pak@brown.edu, mlittman@cs.brown.edu

ABSTRACT
We investigate the practicality of letting average users customize smart-home devices using trigger-action ("if, then") programming. We find trigger-action programming can express most desired behaviors submitted by participants in an online study. We identify a class of triggers requiring machine learning that has received little attention. We evaluate the uniqueness of the 67,169 trigger-action programs shared on IFTTT.com, finding that real users have written a large number of unique trigger-action interactions. Finally, we conduct a 226-participant usability test of trigger-action programming, finding that inexperienced users can quickly learn to create programs containing multiple triggers or actions.

Author Keywords
End-User Programming; Home Automation; Smart Home; Internet of Things; Condition-Action Programming

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION
While the technologies that enable home automation and smart homes have been around for decades, they have been expensive and complex [1]. As a result, smart homes have mostly been embraced by a small community of affluent and technically proficient early adopters [11]. In the last two years, however, companies have begun to market "smart devices" to average consumers at much lower price points, creating momentum toward mass-market pervasive computing.

As a result, the idea of giving an end user tools to program their environment, long discussed in the academic literature [2, 3, 5, 10, 13, 18], is now becoming possible for a larger population. This programming often takes the form "if *trigger*, then *action*," which we term *trigger-action programming*. For instance, the website IFTTT ("If This Then That") enables average users to engage in trigger-action programming with household devices, such as the Philips Hue [15] lights and the Belkin WeMo family of outlets, switches, and motion sensors (Figure 1). Crowd-funded devices from Twine [17] to WigWag [19] have adopted similar paradigms.

CHI 2014, April 26 - May 01, 2014, Toronto, ON, Canada.

ACM 978-1-4503-2473-1/14/04?$15.00.
http://dx.doi.org/10.1145/2556288.2557420

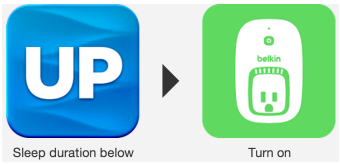

Figure 1. The IFTTT recipe "If I get less than 5 hours of sleep, put on a pot o' coffee" (by IFTTT user kev)

Despite the literature and nascent real-world implementations of trigger-action programming for smart devices, key questions remain. Where should we strike the balance between expressibility and ease of use? For example, IFTTT restricts users to a single trigger and single actions. Do users need more flexibility? Can they handle more flexibility? Does additional programming experience change this equation? Are important elements missing from the model? We investigate these questions through three related studies.

To determine whether trigger-action programming captures smart-home behaviors that users actually desire, we investigate the following hypotheses about smart-home behaviors:

- **H1**: Many desired behaviors for a smart home are expressible using trigger-action programming.
- **H2**: Some behaviors require multiple triggers or actions.

We asked 318 workers on Mechanical Turk (MTurk) to tell us five things they would want a hypothetical smart home to do. Most behaviors they submitted could be expressed naturally using trigger-action programming, confirming prior work [5] with a larger sample. Surprisingly, users' prior programming experience was not significantly correlated with whether their desired behaviors involved programming. Unlike the affordances of IFTTT, 22% of programming behaviors required more than one trigger or action. We highlight three novel levels of abstraction in the triggers participants used, the highest of which is a natural application of machine learning.

To test whether customizability is even necessary for smart-home programming, we investigate the following hypothesis:

- **H3**: In practice, users will combine triggers and actions in a large number of unique ways.

We found many unique combinations among MTurk users' desired behaviors. We also scraped all 67,170 programs end users shared on IFTTT, finding similarly high diversity.

Finally, we conducted a usability test of trigger-action programming to investigate whether trigger-action programming is simple enough for an average user to learn in minutes. We mirrored the interaction design of IFTTT and enrolled a new

set of 226 MTurkers to solve 10 programming tasks. Participants were randomly assigned to use either an interface that supports only a single trigger and a single action (*simple* interface) or one that supports multiple triggers and multiple actions (*complex* interface). We investigate:

- **H4**: Participants perform with equivalent accuracy and speed using either the simple or complex interface.
- **H5**: Prior programming experience and experience using trigger-action programming increase speed and accuracy.

We found no significant differences in participants' performance or satisfaction between the simple and complex interfaces, supporting H4. Furthermore, participants using the complex interface were able to complete complex tasks at a similar rate of success. H5 was only partially confirmed; programming experience did not provide a significant advantage. Participants were slower attempting their first task, yet approached steady-state performance by the second task.

Overall, we found evidence that trigger-action programming with multiple triggers and multiple actions can be a practical approach for smart-home programming. We discuss open questions about triggers requiring interactive machine-learning approaches, as well as how triggers compose.

BACKGROUND AND RELATED WORK

To contextualize our research, we introduce trigger-action programming and provide an overview of related work.

Trigger-action Programming

In trigger-action programming, end users specify the behavior of a system as an event (*trigger*) and corresponding *action* to be taken whenever that event occurs. Both the trigger and the action can contain parameters that customize their behavior. A concrete, instantiated example of trigger-action programming is the website IFTTT, which stands for "if this, then that"—one way of describing the trigger-action pair. In an IFTTT *recipe*, the trigger and action are each selected from a specified *channel* of related behaviors. Recipes can also be shared with the IFTTT user community.

IFTTT has traditionally focused on recipes that link social media sites. For example, the most popular shared recipe notices when a user's Facebook profile picture changes, updating his or her Twitter picture to match. This recipe has been activated by 25,200 other users. In this example, Facebook is the trigger channel and Twitter is the action channel. However, IFTTT supports an increasing number of smart-device channels. Among these channels are blink(1) (USB-controlled indicator light), Philips Hue (light bulbs), Up by Jawbone (wrist-worn pedometer), WeMo Light Switch (wall-mounted switch), WeMo Motion (motion sensor), WeMo Switch (electrical outlet), and Withings (bathroom scale).

To study trigger-action programming in a controlled way, we created our own prototype interface modeled after IFTTT. Figure 2 shows an example of the recipe "Turn on the kitchen lights every day at 6pm" in our interface. Here, Date/Time is the trigger channel and Lights is the action channel. After

Figure 2. Recipe of "If it is 6pm, then turn the lights on."

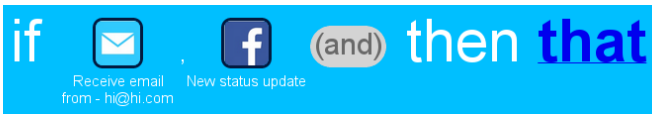

Figure 3. Complex interface with multiple triggers in the recipe

selecting a channel, the user chooses a specific event or action built into that channel. Users can add, edit, and delete channels from their recipe. Our prototype interface included the following channels: Date/Time, Door, Dropbox, Email, Facebook, Fan, Feed, Heater, Light, Motion Sensor, Phone, Switch, Television, Text, Twitter, and Weather.

Although IFTTT offers 69 different channels, the programming paradigm of having a single action triggered by a single event is inherently limited. Therefore, we created two versions of our prototype interface. Whereas our *simple* interface (Figure 2) only allows for one trigger and one action channel per recipe, our *complex* interface (Figure 3) allows users to add multiple trigger and action channels. All triggers are composed by the conjunction "and." For instance, one could trigger an action via "if it is 6:00pm and raining."

Related Work

A number of researchers have investigated end-user programming in smart homes. Newman [12] argues that end-user configurability is key in smart-home applications and advocates sharing insights across a community of users.

Most closely related to our work, Dey et al. [5] interviewed 20 participants about context-aware applications, finding the most common mental model to be rule-based (if-then), as in trigger-action programming. They tested a rule-based prototyping system for context-aware applications, which non-programmers found intuitive, accurate, and expressive. We confirm that many desired behaviors can be expressed using trigger-action programming, and that about one-quarter of these behaviors require the conjunction of multiple triggers. We build upon their work by investigating how the IFTTT community uses trigger-action programming in the real world, as well as comparing the usability of simple and complex trigger-action programming.

Many researchers have proposed smart-home interfaces that share characteristics of trigger-action programming. Truong et al. [18] devised CAMP, which uses a magnetic-poetry metaphor for natural-language expression within a constrained vocabulary. They found that end users were able to express their applications in CAMP, doing so in a way that was people- or task-centric, rather than device-centric. Litvinova and Vuorimaa [10] showed that this idea could be applied successfully with a less constrained interface. Pane et al. [14] also found that an event-based programming style

compatible with trigger-action programming is the most common way for end users to solve problems. Dahl and Svendsen [2] compared several proposals for end-user composition interfaces—a jig-saw interface, a wiring-diagram interface, and a filtered list interface. Their participants found the jigsaw-puzzle interface most fun and engaging, whereas the filtered list interface was the most efficient.

Although these studies validated the concept of trigger-action programming, we investigate open questions. We revisit the premise that trigger-action programming is necessary because of the great diversity of behaviors users will want, evaluate the usability of having multiple triggers/actions versus single triggers/actions, and investigate whether there are learning effects or benefits of prior programming experience.

Trigger-action programming introduces many subtleties. García-Herranz et al. [8] articulate considerations for end-user programming, such as making the distinction between triggers and conditions transparent. Newman et al. [13] built and tested OSCAR, a trigger-action programming interface. Their participants liked the interface and completed a number of tasks, yet had trouble reusing configurations and judging whether they had completed a task accurately. Davidoff et al. [3] note the importance of exceptions and conflicts. In separate work, Davidoff et al. [4] caution that the home is collaborative and conflict resolution is complex.

In contrast, Brush et al. [1] argue that trigger-action programming can be difficult for users to debug when problematic behaviors inevitably occur. Rashidi and Cook [16] suggest using machine learning over end-user programming, while others [9] argue in favor of control using a mobile phone.

STUDY 1

To investigate the practicalities of using trigger-action programming for controlling physical devices, we conducted three related studies. In the first study, we asked workers on Amazon's Mechanical Turk (MTurk) to list five things they would want a smart home to do. As a first-order approximation of trigger-action programming's viability, we analyzed whether these tasks could be expressed through trigger-action programming. In addition, we analyzed the semantics used for triggers and actions. We also counted the number of ways in which different triggers and actions were combined. If the diversity of combinations is high, trigger-action programming or some other approach involving end-user programming could be beneficial. In contrast, if diversity is low, an enumerated list or "app store" might be preferrable to end-user programming.

Methodology

We recruited MTurk workers for a "survey about smart homes and home automation." We asked them to "imagine that you have a home with devices that are Internet-connected and can therefore be given instructions on how to behave. What are five things you would want your home to do?"

On the one hand, we were curious whether participants who were not given further direction would suggest behaviors that could naturally be expressed with trigger-action programming. On the other hand, we feared that participants might suggest very futuristic behaviors, such as flying robot butlers, and were curious whether participants could generate unique and interesting tasks for their home that followed a particular programming paradigm. Therefore, we randomly assigned half of participants to receive no further instructions, and the other half to be primed for trigger-action programming with the following examples based on a promotional video from a home-automation startup [19]: "If I walk down the hall at night, my path should light up. If I leave my house, things should shut down. When I am walking by a sprinkler, I don't want to get wet (and the sprinkler turns back on when I leave). When the mail gets delivered, notify me."

We were also interested in whether the behaviors participants submitted would vary by demographics. Therefore, we collected their age, gender, and whether they have "no experience," "some experience," or "experience with computer programming," or if they are an "advanced computer programmer." We restricted the survey to U.S. workers with a 95% approval rating. We compensated participants $0.45.

Two independent coders first coded each response as *unclear* what the subject's intent was, or into one of the following categories developed collaboratively from pilot-study data:

- **Programming**: A combination of primitive functions that an end user might consider combining in other ways. (e.g., automatically turning on the lights when it is dark outside)
- **Self-regulation**: A house automatically determines a subject's preference and takes action. (e.g., adjust the house to my preferred temperature at all times)
- **Remote control**: End user wants to be able to control devices immediately, not based on a schedule. (e.g., hitting a button on my phone to turn on the lights)
- **Specialized functionality**: Could perhaps be built or programmed, but the behavior is specific to the hardware involved and therefore not amenable to end-user repurposing. (e.g., a breakfast-making machine)

The percentage agreement between the two coders was 89.1% (Cohen's $\kappa = 0.79$). The two coders discussed the disagreements and came to consensus on the code for all items.

The two coders then independently coded each rule's structure. In particular, they coded whether the behavior could be expressed in trigger-action programming with either a *single* or *multiple* triggers, as well as a *single* or *multiple* actions, or whether the behavior required a traditional programming language like Python. Notably, many existing instantiations of trigger-action programming, such as IFTTT, only support a single trigger and a single action.

Using data from the pilot study and from the real study, we collaboratively and iteratively developed a codebook of 21 trigger *channels*, which we defined to include all triggers related to a particular concept (e.g., "weather," "door"). We also developed 46 action channels (e.g., "door"). Within each

trigger and action channel, our codebook included the particular triggers or actions (e.g., "door: open," "door: close"). The 21 trigger channels encompassed 71 particular triggers, while the 46 action channels encompassed 99 particular actions. Note that our codebook includes sensors that do not currently exist, yet were conceivable (e.g., "food: cooked?").

The coders first determined whether each programming example could be represented using trigger-action programming. Then, for all examples of trigger-action programming, they independently proposed a trigger-action program using this codebook. They coded the number of triggers as either single (e.g., "If it is 7:00PM") or multiple (e.g., "If it is 7:00PM and raining')). Similarly, they coded the action as single or multiple. The two coders agreed on the overall structure of 88% of programs (Cohen's $\kappa = 0.87$); they then met to reach consensus and agree on a final program.

We ran regressions to investigate whether age, gender, or programming experience impacted the type of behaviors participants desired for their smart homes. These data were not independent since each participant specified five different behaviors. Therefore, we used mixed models to nest the five behaviors under a particular user. In particular, we created a cumulative-link (logit) mixed model in which the binary dependent variable represented whether or not a particular behavior was coded as programming.

We also analyzed the extent to which different triggers were combined with different actions. We were also interested in estimating the extent to which users might want to combine the given triggers and actions in ways that were not represented by the data. That is, how much of the space of expected combinations is not represented in our current data? In a sense, this question is unanswerable—how can we use the collection of programs to estimate the likelihood of a combination that is *missing* from the collection? Fortunately, this problem has been studied previously in situations like estimating the probability mass of missing words from a corpus of language. The Good-Turing estimate [7] uses the number of singletons in a sample to estimate the probability mass on unseen examples. Roughly, we assume that the trigger-action programs that appear among our data are independently sampled according to user interest in the corresponding behavior.

Participants

We collected five desired smart-home behaviors from 318 MTurk participants, for a total of 1,590 desired behaviors. While our participants ranged in age from 18 to 70, 62.9% of our participants were between 20 and 29 years old. The median age was 25, while the mean age was 28.2 ($\sigma = 9.1$). Our sample was 69.2% male and 30.0% female; 1.3% of participants declined to state their gender.

Our sample was also biased towards individuals with computer-programming experience, possibly because MTurk workers with interest in technology would be more interested in completing a task about "smart homes." Whereas 117 participants (36.9%) had no computer-programming experience, 103 (32.5%) had some programming experience, 84 (26.5%) had programming experience, and 13 (4.1%) were advanced.

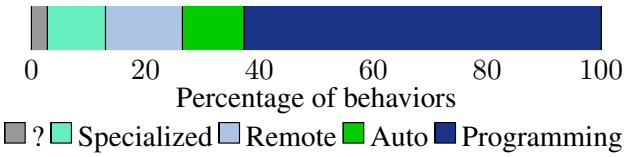

Figure 4. The percentage of behaviors submitted by our MTurk participants that were unclear (*?*) in intent, required *specialized* functionality unavailable in current homes, used a smartphone or similar device as a *remote* control for household devices, involved *automatic* self-regulation, or required *programming*.

Results

Overall, we found that the majority of the five smart-home behaviors desired by each of our MTurk workers involved programming. Furthermore, all of these programming behaviors could be expressed using trigger-action programming as long as the paradigm supports multiple triggers and actions.

Requests often involve programming

We found that programming does indeed capture most of what participants immediately thought of when asked what they wanted a hypothetical smart home to do. As described in the methodology, we coded each behavior submitted by our participants as "programming," "remote control," "specialized feature," automatic "self-regulation," or "unclear." Figure 4 depicts the proportion of behaviors in each category.

A total of 62.6% of the behaviors our participants submitted centered on programming, highlighting the importance of providing average users with the tools they need to program their smart home. Note that we viewed programming as the notion of specifying behaviors that, given triggers and actions that could be specified at the appropriate level of abstraction, could occur now or in the future. For example, we coded all of the following examples as programming:

- "I want the fan in my room to turn on when it is hot."
- "Notify me if my pet gets out of the backyard."
- "Start brewing coffee 15 minutes before my alarm."
- "Lights...dim according to the level of outside light."
- "I would like my home to automatically clean the floors on a daily basis while no one is in the room."

The most popular non-programming request, representing 13.5% of the behaviors submitted by our participants, was "remote control" functionality. This type of request involved using a smartphone, computer, voice, or a gesture to control a household device at the current moment. In particular, the primary distinction between the "remote control" and "programming" categories is that "programming" involves specifying a behavior that could be activated in the future, rather than right now. "Remote control" requests included "start the coffee pot from my bedroom," "lock and unlock the doors on command with your voice," and "change [the] temperature of each individual room via phone."

Another 10.8% of responses required "specialized functionality" that does not currently exist. Examples in this category included "pet my cat for me," "YouTube on bathroom mirror," "do my hair for me," and "I want my mail to be delivered

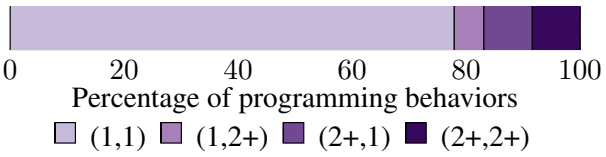

Figure 5. The distribution of programming structures required to express the programming behaviors our participants submitted. Each of the four categories can be expressed using a variant of trigger-action programming. Category names refer to the number of triggers and number of actions, respectively. For example, the conjunction of multiple (2+) triggers causing a single action is coded as (2+,1).

by a robot." Neither an average user nor an experienced programmer would be able to program these tasks except through an extraordinary investment of time. A further 10.1% of responses involved "self-regulation," such as temperatures that adjust automatically to the perfect temperature. We coded the final 2.8% of behaviors, like "smart lights," as "unclear." We were unable to determine the intent of those submissions.

Trigger-action programming structure

Surprisingly, none of the desired behaviors would require a full programming language to implement as long as sufficiently expressive triggers and action exist. To understand whether multiple triggers and multiple actions should be supported in trigger-action programming, we coded the structure of a trigger-action program using the triggers and actions in our codebook. We excluded the 37.4% of behaviors identified in the previous analysis as non-programming.

A total of 77.9% of the programming behaviors fit into the single trigger, single action construct used by IFTTT. Examples of this type included "close the blinds when the sun is too bright" and "call to let me know when the kids get home."

Although 77.9% of programming behaviors could conceivably be expressed with a single trigger and single action if given appropriate triggers and actions, the remaining 22.1% required either multiple triggers or multiple actions. While 5.2% of behaviors required only a single trigger with multiple actions, 8.5% required multiple triggers for a single action. The latter type often captured things that should occur only in particular circumstances. For example, "If there is a package delivered and I'm away, I'd like to be notified via email." Finally, 8.4% of behaviors required both multiple triggers and multiple actions. For instance, one participant specified, "When I get up at night, I would want my lights to turn on and off as I enter and exit the room."

Expressiveness by demographic

Although we expected participants with programming experience would be more likely to submit programming behaviors, this was not the case ($p = .226$). Furthermore, neither the participant's gender ($p = .487$) nor age ($p = .407$) was a significant predictor of whether a behavior was programming.

As described earlier, half of the participants were primed with examples of trigger-action programming behaviors. Whether the participant saw these examples was a significant factor ($p < .001$) in whether the submitted behavior was programming. Whereas 51.0% of behaviors submitted in the absence of examples represented programming, 68.9% of the submissions from participants who saw the examples were programming. This result suggests users can be primed to some degree to think of programming behaviors for smart homes. However, even without examples, the majority of behaviors were still programming, suggesting that end-user programming can implement many behaviors desired of smart homes.

Triggers' level of abstraction

To understand the types of triggers that trigger-action programming should support, we examined the level of abstraction at which participants articulated, or directly implied, triggers. We collaboratively identified three distinct levels of abstractions among the 71 different triggers in our codebook.

As in prior work [5, 18], we found that participants tended not to mention sensors directly. For instance, instead of discussing a motion sensor being activated, participants would specify that an action occur when someone walks into a room. Nonetheless, 31 different triggers were indeed sensors in the engineering sense, which we identified as the lowest level of abstraction. Common triggers in this category included the state of a device changing (e.g., a doorbell ringing, a device being turned off), dates/times, moisture, sound, and light.

The second category represented activities, locations, and states similarly abstracted from physical sensors. For example, one participant explained, "I want the sink to turn on when I pick up my toothbrush." While one cannot purchase a "picked-up-ness" sensor, the activity of picking up an object could be inferred from an accelerometer. Overall, our codebook contained 26 different triggers at this abstract level. Triggers related to occupancy and location were the most common. For example, one participant stated, "I would like my home to automatically clean the floors on a daily basis while no one is in the room." Current and forecasted weather conditions were also very common. Additional triggers in this category included individuals waking up, mail being delivered, and objects not being used for a period of time.

The third category, which we term *fuzzy triggers*, comprises 14 triggers whose implementation would require a number of questions to be answered. These triggers require complex decision making. One such trigger centers on anomalous events or other deviations from normalcy, such as, "I would like to be notified when my pool chemicals drop lower than normal." This trigger would require a sense of what is normal, which may vary by person, time, or other factors. Other fuzzy triggers involved detecting when a person or pet was hungry, when food had cooked sufficiently, when an area had become dirty, or when someone was uncomfortable.

Fuzzy triggers likely involve machine learning. For instance, interactive reinforcement learning could be used to determine what household conditions a particular person deems uncomfortable. Given that participants commonly used semantics at this level of abstraction, a successful interface for trigger-action programming should likely support fuzzy triggers.

Diversity of behaviors

End-user programming in the home would be primarily advantageous if users want to define a large number of unique

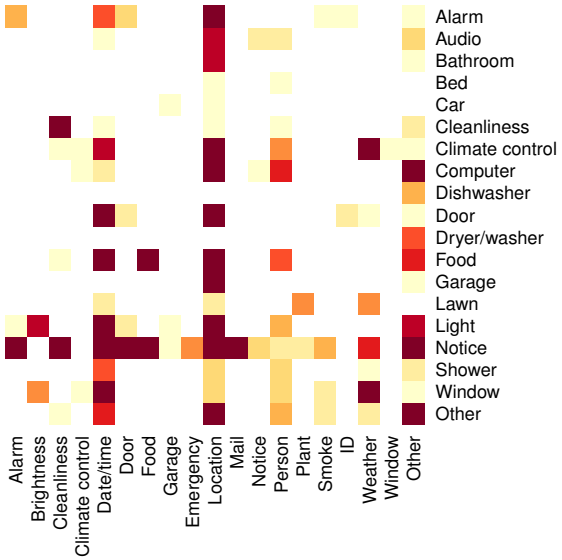

Figure 6. A heatmap showing how participants' desired smart-home behaviors combined trigger channels (x axis) and action channels (y axis).

behaviors. We analyzed the diversity of the 995 behaviors classified as programming. We first explore how triggers were combined and separately how actions were combined. Then, we examine diversity among the complete behaviors.

Participants used the 21 different trigger channels we coded (e.g., "location," "weather") in 47 distinct combinations. For example, the single trigger "location" represented one combination, while using "date/time" and "weather" together to trigger a behavior represents another distinct combination. Using the Good-Turing method, we estimate a missing probability mass of 2% of possible trigger combinations. In other words, if another MTurker submitted a behavior, with 98% probability we would expect it to be one of the 47 combinations we saw in our previous data. With 2% probability, however, we would expect it to be a trigger channel or combination of trigger channels we did not observe in our data.

Similarly, participants used the 46 action channels in 77 distinct combinations among the 995 programming behaviors. Using Good-Turing, with 4% probability we expect an additional behavior that is submitted to use a hitherto unseen action channel or combination of action channels.

Among the 995 programming behaviors were 236 distinct combinations of trigger channels and action channels. Figure 6 shows the frequencies of these combinations; for visual clarity, the heatmap combines related channels. The Good-Turing estimate suggests that 13% of the probability mass is missing from our data. In other words, with 13% probability, the next behavior submitted would be a combination of trigger channel(s) and action channel(s) we had not yet seen.

We also looked beyond the channels (e.g., "Weather") to the triggers and actions themselves (e.g., "Weather: Starts to Rain"). Among the 995 programming behaviors, we observed 464 distinct programs based on our codes. Notably, 316 of the programs appeared only once in our data. The diversity of programming behaviors among the first five desired behaviors our MTurkers submitted suggests that end-user programming in smart homes could be useful.

STUDY 2

Although end-user programming for home automation is not yet widespread, early adopters on the website IFTTT already use trigger-action programming in the real world. We scraped all programs shared publicly on IFTTT to better understand the combinatorics of trigger-action programming in practice.

Methodology

We downloaded all 67,169 *recipes* (trigger-action programs) that had been shared publicly on `IFTTT.com` as of June 20th, 2013. Our scrape does not include recipes users created but did not actively choose to share publicly. Each recipe consists of an "if channel" and its trigger condition and any parameters, as well as a "then channel" and its selected action and any parameters. Each recipe includes a recipe id, an author id, and statistics on when it was shared and how many other users have activated it. Although some recipes shared on IFTTT involve physical devices, many do not. We downloaded all public recipes, yet focused on the six channels for controlling physical devices: blink(1), Philips Hue, Up by Jawbone, WeMo Motion, WeMo Switch, and Withings.[1]

We analyzed which triggers were associated with which actions. As in Study 1, we used Good-Turing estimation to quantify the probability of combinations *not* represented in the sample. In the case of IFTTT, the Good-Turing assumption of independence of samples is not valid. A recipe that is shared on IFTTT is less likely to be added again by another user; that user could simply activate the existing recipe. Nevertheless, duplicate recipes appear frequently among our data, so we feel the independence assumption is sufficiently accurate to serve as the basis of a useful measure.

Results

Although IFTTT recipes can be created by average users, shared recipes can be treated as an "app store" that allows users to browse and activate other users' recipes. Our scrape includes 67,169 shared recipes written by 35,295 different authors. Individual authors contributed between 1 and 126 recipes (32% contributed 2+). Individual recipes had been activated by between 0 and 25,200 other users (median 1).

We focus on recipes involving the aforementioned six physical devices. These six devices support 16 different trigger events and 18 different actions. Among the shared recipes in our scrape, the device channels were paired with 44 different other channels. For example, one recipe paired "Date/Time" with the Philips Hue to create a recipe its author described as "turn off lights mid day." However, only 513 recipes in our scrape (0.8%) use physical devices as triggers, while only 858 (1.3%) use physical devices as actions. In 92 cases, the recipe combined a device-oriented trigger with a device-oriented action, creating a "pure" device recipe. Figure 7 shows the frequency of each of these combinations as a heatmap.

[1]IFTTT has since added support for additional physical devices, most notably SmartThings (motion and presence sensors, windows, locks, and outlets), Google Glass, and the WeMo Light Switch.

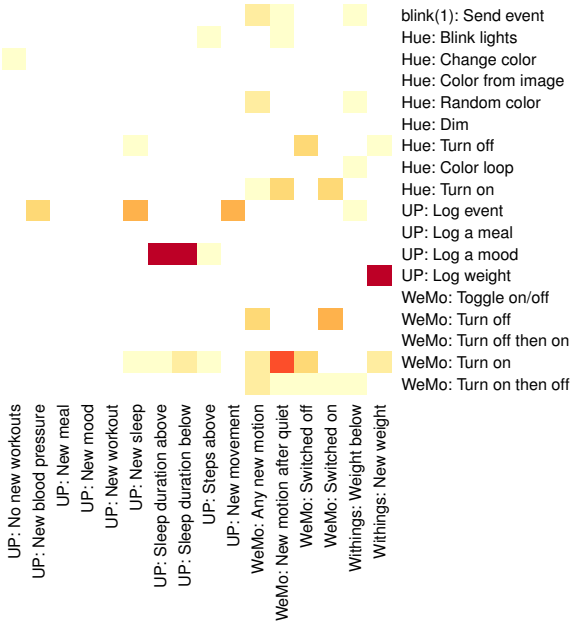

Figure 7. A heatmap showing the frequency of IFTTT recipes combining physical triggers with physical actions.

A total of 513 recipes used a physical device as a trigger. Of 368 possible pairings of these 16 physical triggers with 23 different physical or virtual action channels, 147 unique combinations appeared (40%). A total of 858 recipes in the recipe database used a physical device as an action. Of the 594 possible pairings of these 18 physical actions with 33 distinct trigger channels, 176 unique combinations appeared (30%).

Among recipes with physical triggers or physical actions, the most popular dozen channels were liberally combined with other channels, suggesting that users might find it easier to specify a meaningful combination than search for the one they want in a long list. Good-Turing estimation predicts the probability of generating a combination that did not appear in this sample is approximately 11% for the device triggers and approximately 9% for the device actions. This result implies that roughly 1 in 10 searches would come up empty if users were constrained to only use recipes in this scrape, suggesting that even treating the vast set of existing IFTTT recipes as an "app store" would not cover all desirable combinations.

STUDY 3

Our final study investigated the usability of trigger-action programming, as well as how demographic factors and support for multiple triggers/actions correlate with usability.

Methodology

We recruited MTurk workers for a "research study on technology for the home." Participants were compensated $2.00 for the study, which took about 30 minutes. We randomly assigned each participant to use either an interface that supported a single trigger and single action (*simple* interface) or an interface that was identical except for supporting multiple triggers and multiple actions (*complex* interface), as described in the background section. We measured the time a participant spent on a task and whether their program was correct.

Participants answered demographic questions, used the assigned interface to attempt the ten tasks listed in Table 1 (assigned in random order), and completed a final Likert-scale satisfaction survey. Either interface could be used to solve Tasks A–F, while only the complex interface could be used to solve Tasks I–J. Task G and Task H were impossible to solve within the constraints provided. We informed participants that they should click a "skip task" button if they were unsure how to solve a task or believed it impossible.

We ran regressions to analyze participant's success rates and timing. As these data were not independent, we used mixed models. For success solving the task, which is a binary outcome, we created a cumulative-link (logit) mixed model. For the time it took to solve a task, which is continuous, we created a linear mixed model. We used six explanatory variables: the task that was being attempted, the number of tasks already attempted, and the interface (simple vs. complex), as well as the participant's gender, programming experience, and age. We also included four hypothesized interaction terms.

We analyzed participants' Likert-scale responses to satisfaction questions using ordinal logit regression, corrected for multiple testing using the Bonferroni method. Our independent variables were the interface the participant saw, as well as the participant's gender, age, and programming experience.

	Instruction
A	When my Facebook profile picture changes, update my Twitter profile picture.
B	Get all updates from the website www.xkcd.com via email.
C	If I receive an email from JohnDoe@gmail.com then blink lights to notify me.
D	Turn on the lights when the sun sets.
E	If it is 7:00PM then turn on the lights in my bedroom.
F	Blink the lights if someone is at my front door.
G	The lighting in my bedroom should be on when I am there and off when I am not there.
H	If it begins to rain then change the light colors to blue.
I	If it is 10:00pm and my bedroom door is closed and the lights are off, turn the television off.
J	When I close the kitchen door, lock the door and turn off the kitchen light.

Table 1. The ten tasks in our usability study, which were assigned in random order. Tasks G and H were impossible to solve, while Task I and J could only be solved using the complex interface.

Participants

We had 226 participants; 107 were male (47.4%), 118 were female (52.2%), and one declined to answer (0.4%). Participants hailed from 45 different U.S. states and ranged in age from 18 to 67 (median = 30, mean = 32.6, σ = 10.9). Whereas 163 participants (72.1%) did not have any programming experience, 31 (13.7%) had some programming experience, 23 (10.2%) had programming experience, and 9 (4.0%) considered themselves advanced programmers.

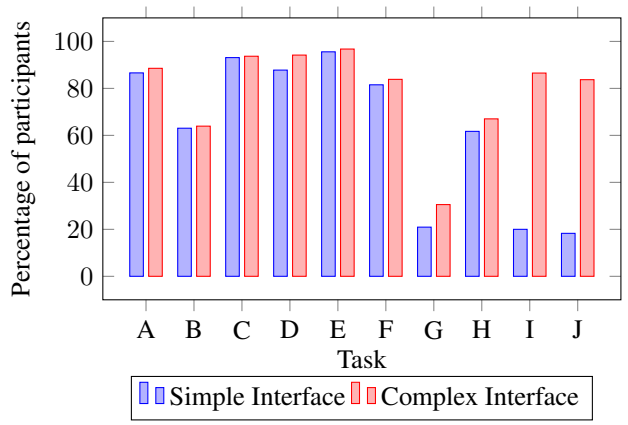

Figure 8. The percentage of participants who successfully completed each task. Tasks G and H were impossible to solve using either interface, as were Tasks I and J for participants using the simple interface. Success in these cases is defined as a participant skipping the task.

Only five participants (2.2%) had previously used IFTTT (two rarely and three occasionally). Participants were relatively tech-savvy; 89% own a smartphone, 63% own a tablet, 10% own a "smart thermostat," 9% own a "robotic vacuum cleaner," and 3% own a "home-automation device."

Results

Overall, participants successfully used either the simple or complex interface to create programs. We observed a learning effect in which participants' performance quickly improved as they completed successive tasks. Some tasks were more difficult than others, and older participants performed worse.

Task completion

Participants successfully completed most tasks. As shown in Figure 8, most tasks were solved by 80% or more of participants. Relative to Task A, the control in our cumulative-link (logit) mixed model, participants were significantly less likely ($p < .001$) to solve Task B: "get all updates from the website `www.xkcd.com` via email." This task required participants locate the "Feed" channel (RSS logo) and then enter the site's URL. We hypothesize that participants' unfamiliarity with RSS caused difficulty. In contrast, participants were more likely ($p = .048$) to successfully solve Task E: "If it is 7:00PM then turn on the lights in my bedroom."

Participants were also less likely to solve Task G ($p < .001$) and Task H ($p < .001$) than Task A. Both tasks were impossible, and we had instructed participants to skip impossible tasks. Only 26% of participants overall correctly skipped Task G, yet 64% of participants skipped Task H. Task G, "The lighting in my bedroom should be on when I am there and off when I am not there," was impossible because it required either the existence of an occupancy sensor or the use of two programs (triggered, respectively, on entry and exit), which neither interface supported. Task H, "If it begins to rain then change the light colors to blue," was impossible because there was no option to change the color of a light, which we hypothesize was more obvious to participants.

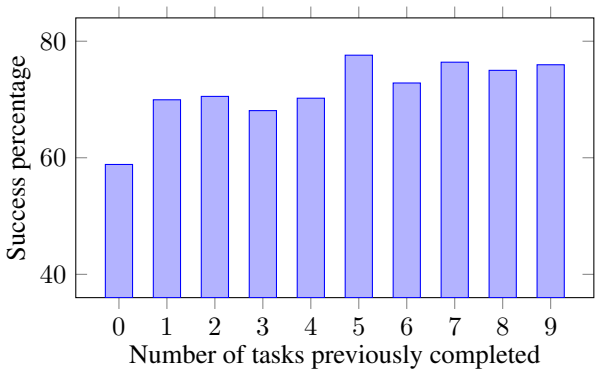

Figure 9. The percentage of participants who successfully completed a task versus the number of tasks they had previously completed. We found a learning effect; participants got better at solving tasks over the ten tasks they completed.

Tasks I and J could only be solved using the complex interface. Success for the simple interface reflects the proportion of participants who skipped the task as instructed. Unsurprisingly, participants who used the simple interface were significantly less likely to correctly solve Tasks I and J ($p < .001$).

Across all tasks, we observed a learning effect, partially supporting H5. The number of tasks previously attempted was significantly correlated with success solving the current task ($p < .001$), as shown in Figure 9.

Older participants less success overall ($p = .040$). In contrast, participants' gender ($p = .931$), prior programming experience ($p = .497$), and whether they used the simple or complex interface ($p = .794$) were not significantly correlated with success at solving tasks that could be solved with both interfaces (Tasks A–F) or realizing that certain tasks could not be solved (Tasks G–H). Furthermore, we did not observe any significant interaction effects between the task and prior programming experience, between the number of tasks completed and prior programming experience, or between the number of tasks completed and the interface participants saw.

Time to complete a task

We also investigated how long it took the participant to solve Tasks A–F, solvable using either interface. We created a linear mixed model with the time (in ms) from loading the interface to submitting the solution as the dependent variable. We excluded data from incorrect solutions and tasks that were impossible with the assigned interface.

The participant's age and the number of tasks they had already attempted impacted the completion time. Older participants took longer to complete these tasks ($p < .001$). As shown in Figure 10, participants got faster as they completed more tasks ($p = .003$), yet seemed to reach a steady state by their second task. Relative to Task A, participants took significantly longer to complete Task B ($p < .001$). Notably, prior programming experience ($p = .214$), gender ($p = .756$), and the interface ($p = .905$) were not significant factors.

Satisfaction

Overall, participants appeared satisfied with the usability of both the simple and complex interfaces. At the end of the

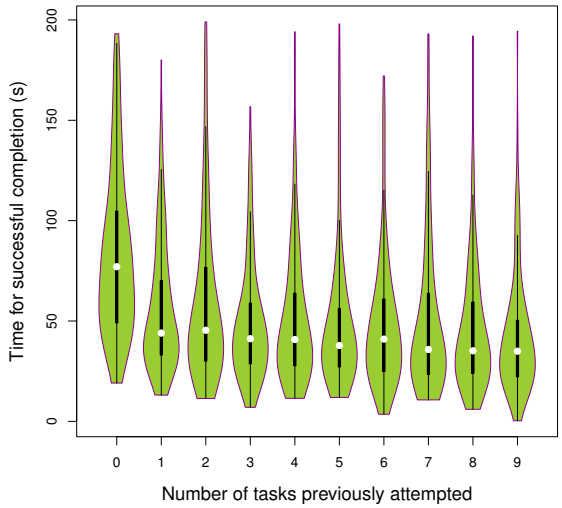

Figure 10. A violin plot (combination box plot and kernel density plot) of the time it took participants to solve a task versus the number of tasks they had previously attempted. We exclude unsuccessful solutions.

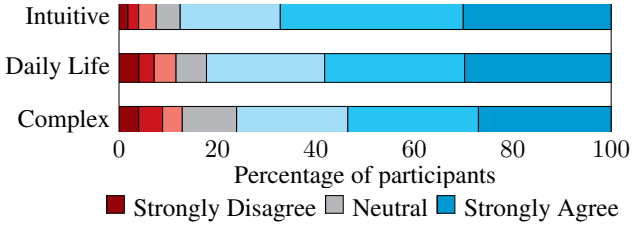

Figure 11. Participants' responses on a 7-point Likert scale to the following statements: "Overall, it was easy and *intuitive* to create recipes"; "I would be interested in creating recipes of this sort in my *daily life*"; "I believe I could handle a more *complex* programming interface."

study, participants responded on a 7-point Likert scale to three statements about the interface. As shown in Figure 11, over three-quarters of the participants agreed that "it was easy and intuitive to create recipes," that they would be "interested in creating recipes of this sort in daily life," and that they thought they "could handle a more complex programming interface."

For each statement, we performed ordinal logistic regression with participants' Likert-scale responses as the dependent variable. Unsurprisingly, participants with prior programming experience were significantly more likely to agree that they could handle a more complex programming interface ($p < .001$). Neither the interface used nor the participant's age and gender were significant factors.

DISCUSSION

We have provided evidence that average users can successfully engage in trigger-action programming with multiple triggers and actions. Furthermore, this paradigm can express many of the smart-home behaviors that immediately came to our participants' minds, and both hypothetical and real-world examples of trigger-action programming exhibit substantial diversity. However, scaling trigger-action programming for use in practical settings introduces a number of challenges.

While coding participants' desired behaviors and designing the usability study, we found that the semantics of how multiple triggers compose with each other are complicated. Triggers could conceivably contain an *event*, a *condition*, or some combination. Some objects, like a door, would need to support an event ("the moment the door closes"), a condition ("is the door closed?"), and an action ("close the door"). Notably, two events do not compose well. For example, the exact moment the temperature drops below freezing is unlikely to be the exact moment the window is opened.

Instead, we found that a trigger generally should contain exactly one event, optionally alongside multiple conditions. For instance, "heat the floors [action] during the winter [condition] down the path I'm walking [event]." Triggers that seem to contain only conditions, like "notify me if my windows are open past 10pm," could be expressed as checking every X minutes between 10:00pm and the morning (event) whether the window is open (condition). However, one could imagine many subtly incorrect ways to write this program, such as to check at precisely 10:00pm (event) if any windows are open (condition), which would miss when someone opens a window after 10:00pm. The usability of these distinctions requires substantial future investigation as part of a field study.

The level of abstraction participants used to express triggers also suggests substantial future work. A handful of participants used triggers that were direct sensors (e.g., "motion is detected"), which are the simplest to implement. As in past work [5], many participants expressed triggers one level of abstraction higher (e.g., "when I am in the room"), which suggests that the HCI community and sensor network communities could collaborate to enable interface affordances that match users' mental models. The third level of abstraction (e.g., "the water is too hot," where "hot" is ambiguous or person-dependent) suggests research opportunities for studying how to create sensors that tailor themselves to the preferences of users based on interactive machine learning.

In the smart home, machine learning could potentially also resolve conflicts more intelligently. Proposals in the literature include giving precedence to more recently created rules or even recipes created by more authoritative users [3]. Instead, given an appropriate metric, one might want to prefer more specific recipes to more general ones. Perhaps better, the system could preemptively identify conflicts and ask the user for resolution either at the time of conflict or when the rule is created. Machine-learning approaches could also interpolate between conflicting actions, mediating conflicts internally.

The place of trigger-action programming within end-user programming for the home deserves consideration. On the one hand, very common or very complex behaviors might best be captured by an "app store," as others have proposed [6]. On the other hand, we found great diversity in the relatively straightforward compositions specified in MTurkers' desired behaviors and IFTTT users' recipes. If an app store becomes too vast, users might find it more efficient to use trigger-action programming than to determine what combination of search terms, if any, leads them to the desired program. Furthermore, advanced users may desire additional concepts from

programming language theory; these features could be hidden from average users. For instance, trigger-action programming does not make it easy to specify that the artificial light in a home adjust to be inversely proportional to the natural light outside. Most likely, a hybrid of all of these approaches will best support both beginner and expert users.

Limitations

While our three studies provide evidence that trigger-action programming can be useful and usable, they had a number of limitations. Most notably, we did not investigate any competing approaches to end-user programming. As a result, we are unable to evaluate the relative strengths and weaknesses of trigger-action programming, nor can we make strong claims that trigger-action programming is the best way forward.

Our data is not necessarily representative of all users. The IFTTT community is a self-selected pool of early adopters. IFTTT and MTurk users are likely more technical than the average person. Neither group is a sample of people who would consider living in a smart home. Furthermore, the things a novice user expects to want a hypothetical smart home to do in the future is far from a perfect proxy of reality.

In addition, we only asked MTurkers to write the first five behaviors that came to mind, rather than all desired behaviors. Actually living in a smart home might further influence users' desires, potentially in unexpected directions. As a result, we cannot claim that trigger-action programming captures a particular fraction of all desirable smart-home behaviors. We plan to conduct a field study that investigates trigger-action programming in a more ecologically valid setting. The role of triggers and sensors based on machine learning (e.g., "comfort" sensors) will be particularly interesting in the field.

A field study can elucidate the impact of subtle issues we have discussed. For instance, conflicts and exceptions are best investigated in the field. Similarly, users' mental models of how to construct a program deserve further study. While a trigger implicitly containing an "or" relationship can be captured by two independent programs with single triggers, average users may not realize how to do so. We also hope to investigate the natural level of abstraction for each type of trigger.

ACKNOWLEDGMENTS

We are grateful to the Rutgers UPOD group, especially Phillip Quiza, and the Brown UPOD group, especially Steve Reiss, for helping us work toward end-user programming of devices. We thank Brown University and the National Science Foundation for financial support.

REFERENCES

1. Brush, A. B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., and Dixon, C. Home automation in the wild: Challenges and opportunities. In *Proc. CHI* (2011).
2. Dahl, Y., and Svendsen, R.-M. End-user composition interfaces for smart environments: A preliminary study of usability factors. In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*. 2011, 118–127.
3. Davidoff, S., Lee, M. K., Yiu, C., Zimmerman, J., and Dey, A. K. Principles of smart home control. In *Proc. Ubicomp* (2006).
4. Davidoff, S., Lee, M. K., Zimmerman, J., and Dey, A. Socially-aware requirements for a smart home. In *Proc. ISIE* (2006).
5. Dey, A. K., Sohn, T., Streng, S., and Kodama, J. iCAP: Interactive prototyping of context-aware applications. In *Proc. Pervasive* (2006).
6. Dixon, C., Mahajan, R., Agarwal, S., Brush, A., Lee, B., Saroiu, S., and Bahl, P. An operating system for the home. In *Proc. NSDI* (2012).
7. Gale, W. A., and Sampson, G. Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics 2* (1995), 217–237.
8. García-Herranz, M., Haya, P., and Alamn, X. Towards a ubiquitous end-user programming system for smart spaces. *Journal of Universal Computer Science 16*, 12 (2010), 1633–1649.
9. Koskela, T., and Väänänen-Vainio-Mattila, K. Evolution towards smart home environments: Empirical evaluation of three user interfaces. *Personal Ubiquitous Comput. 8*, 3–4 (July 2004), 234–240.
10. Litvinova, E., and Vuorimaa, P. Engaging end users in real smart space programming. In *Proc. Ubicomp* (2012).
11. Mennicken, S., and Huang, E. M. Hacking the natural habitat: An in-the-wild study of smart homes, their development, and the people who live in them. In *Proc. Pervasive* (2012).
12. Newman, M. W. Now we're cooking: Recipes for end-user service composition in the digital home. Position Paper– *CHI 2006 Workshop IT@Home*, 2006.
13. Newman, M. W., Elliott, A., and Smith, T. F. Providing an integrated user experience of networked media, devices, and services through end-user composition. In *Proc. Pervasive* (2008).
14. Pane, J. F., Ratanamahatana, C. A., and Myers, B. A. Studying the language and structure in non-programmers' solutions to programming problems. *Int. J. Human-Computer Studies 54*, 2 (2001), 237–264.
15. Philips. Hue. `https://www.meethue.com`, 2013.
16. Rashidi, P., and Cook, D. J. Keeping the resident in the loop: Adapting the smart home to the user. *IEEE Transactions on Systems, Man, and Cybernetics—Part A 39*, 5 (2009), 949–959.
17. Supermechanical. Twine. `http://supermechanical.com/twine/`, 2013.
18. Truong, K. N., Huang, E. M., and Abowd, G. D. CAMP: A magnetic poetry interface for end-user programming of capture applications for the home. In *Proc. Ubicomp* (2004).
19. Wigwag. Wigwag. `http://www.wigwag.com/`, 2013.