THE UNIVERSITY OF CHICAGO

TOWARDS INTEGRATION OF EMERGING TECHNOLOGIES FOR END-USERS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES DIVISION
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
VALERIE ZHAO

CHICAGO, ILLINOIS
FEBRUARY 8, 2023

TABLE OF CONTENTS

ABSTRACT

Technology moves fast, often leaving behind people who do not have technical expertise. For emerging technologies like smart homes and social robots, obstacles to their wide adoption include limited information on their applicability, as well as a lack of end-user support for customization. This dissertation proposes to bridge these gaps in end-user adoption. Specifically, we propose to 1) introduce such technologies to existing systems and examine how they affect user interactions, and 2) design user interfaces that introduce new channels of information to make such technologies accessible to end-users. The technologies we consider include smart home devices, social robots, and body-actuating devices.

To help end-users effectively automate smart home devices and online services, we designed a tool that compares trigger-action programs, which are popularly used in these areas. To ease end-users into reinforcement learning, we designed interfaces that help them define reinforcement learning tasks. Expanding on the existing literature of social robot applications, we studied how introducing humanoid robots to livestreamed performances might foster engagement from performers and from the audience. Finally, to encourage adoption of body-actuating devices like electrical muscle stimulation, we are currently designing interactions leveraging unique features of social robots to engender user trust in these devices.

CHAPTER 1

INTRODUCTION

Technology is evolving from traditional computers, smartphones, and tablets. Now there are smart homes in which appliances share information through the Internet, and augmented and virtual reality systems that make users experience alternative worlds. Although less ubiquitous, there also exist technologies such as social robots that leverage their physical anthropomorphic features to foster social interactions, and body-actuating devices that leverage the human body to enhance one's physical capabilities or communicate information. We anticipate these emerging technologies to permeate all aspects of people's daily lives in the future, from their physical interactions with the environment, to home and work, to education, healthcare, and entertainment. For this dissertation, we propose working towards realizing this vision by integrating these emerging technologies into the lives of end-users – people who do not necessarily have the technical background to overcome the technical complexities of such devices.

For emerging technologies, various obstacles stand in the way of their wide adoption. For technologies such as smart homes and social robots, where we anticipate every household to potentially own in the future, one obstacle is having accessible means of end-user customization. On the one end, end-user programming paradigms enable users to exert more control over the customization without the complexity of software programming languages (e.g. [88, 38, 72]). However, they are not infallible against reasoning errors, resulting in unexpected behaviors in users' programs and subsequent reluctance to adopt these systems (e.g. [9]). On the other end, machine learning methods afford a less demanding and more natural interaction with the technology than end-user programming (e.g. [1, 36, 4]). This is particularly true in the case of social robots, as programming them could negate one's suspension of belief in their social agency. Yet, defining the tasks to learn in the first place can still be nontrivial for end-users, requiring technical knowledge to minimize the cost and maximize the efficacy of the learning.

Another obstacle is the limited knowledge on the potential applications of such technology. Social robots are one such domain. Existing work has found that robots not only offer functional utility—assisting tasks such as manufacturing and engineering—but also offer benefits through social means. Robot interactions can be designed to foster compliance (e.g. [6, 58]) and trust (e.g. [81]), leading to applications such as child tutoring (e.g. [75, 56]) and home assistance (e.g. [69]). However, relatively little work has explored their application in other domains, such as in entertainment and in tandem with other emerging technologies.

To alleviate these challenges for adopting emerging technologies, we propose to 1) introduce such technologies to existing systems and examine how they affect the user interaction, and 2) introduce new channels of information to make such technologies accessible to end-users. This dissertation will include the following projects. First, in the realm of smart homes and end-user programming, we designed a tool that compare trigger-action programs to help end-users iteratively create and debug their smart home automations. Second, we designed user interfaces to help people define reinforcement learning tasks, with a focus on human-robot interaction. Third, we examined the potential of introducing humanoid robots into livestreamed performances for fostering audience and performer engagement. Finally, we will investigate the potential for humanoid robots to engender user trust in body-actuating devices, namely electrical muscle stimulation. The first three projects are complete, while the last one is ongoing.

CHAPTER 2
UNDERSTANDING TRIGGER-ACTION PROGRAMS THROUGH NOVEL VISUALIZATIONS OF PROGRAM DIFFERENCES

2.1 Introduction

Trigger-action programming (**TAP**) is a paradigm for end-user development and composition in which users create **rules** of the form "IF [trigger] WHILE [conditions] THEN [action]" using a graphical interface. For example, the rule "IF Alice falls asleep WHILE it is nighttime AND the front door is unlocked THEN lock the front door" instructs the home to lock the front door when Alice falls asleep at night. We term a set of TAP rules a TAP **program**. TAP has shown promise in empowering non-technical users to connect and automate Internet-of-Things devices and online services [88]. It underpins end-user automation in domains including social media [82], business [93], scientific research [13, 14], and smart homes [89, 55, 74]. Services that support TAP include IFTTT [32], Microsoft Flow [61], Zapier [93], and Mozilla WebThings [60, 20].

TAP is intuitive and easy to use [88, 22], but it is also vulnerable to reasoning errors [30, 92, 9]. Complex interactions between rules and device states can hinder users from knowing precisely when the TAP system would initiate an action [92, 9]. Increasingly large deployments will exacerbate TAP complexity by requiring many rules to satisfy potentially conflicting requirements. This complexity makes it challenging for users to write rules that match their intent or to debug programs, leading to problems ranging from discomfort to wasted resources to security risks.

During the development and maintenance of TAP programs, comparing similar TAP programs is a common and crucial task. We use the term **variants** to refer to highly similar programs being compared. When a user is iteratively tweaking or debugging their program

Program A:

1. **IF** Alice falls asleep **THEN** lock the front door.
2. **IF** it becomes nighttime **THEN** lock the front door.
3. **IF** the front door unlocks **WHILE** Alice is asleep **AND** it is nighttime **THEN** lock the front door.

Program B:

1. **IF** Alice falls asleep **THEN** lock the front door.
2. **IF** it becomes nighttime **THEN** lock the front door.
3. **IF** the front door unlocks **WHILE** Alice is asleep **THEN** lock the front door.
4. **IF** the front door unlocks **WHILE** it is nighttime **THEN** lock the front door.

Program C:

1. **IF** Alice falls asleep **WHILE** it is daytime **THEN** lock the front door.
2. **IF** it becomes nighttime **THEN** lock the front door.
3. **IF** the front door unlocks **WHILE** Alice is asleep **THEN** lock the front door.
4. **IF** the front door unlocks **WHILE** Alice is awake **AND** it is nighttime **THEN** lock the front door.

Figure 2.1: A running example. Programs A, B, and C all have different rules, but only Program A behaves differently from the others. Program A allows the front door to unlock and stay unlocked when Alice is awake at night or when she is asleep during the day. Meanwhile, Programs B and C both ensure the front door remains locked while Alice is asleep and/or it is nighttime.

either to add new behaviors or to fix bugs, the original program and the tweaked programs the user creates are variants. Because TAP programs can be long (contain many rules) or be complex in the degree to which different rules act on the same device, users might modify a program and struggle to determine whether they have achieved their goal. Similarly, taking advantage of ecosystems of shared TAP programs [32], users might merge programs created by others into their own existing programs, wondering how the variant reflecting the combined programs compares to the original. Furthermore, researchers have recently created tools for automatically synthesizing TAP programs [95, 94, 41]. These methods often output multiple candidate variants from a single input, leaving the user wondering which variant to choose.

We further illustrate instances where users may wish to compare TAP programs with the following vignette. Suppose Alice uses Program A (Figure 2.1) for her home. She expects the door to be locked whenever she is asleep or whenever it is nighttime. However, she sometimes notices that the door is unlocked at night, making her concerned about the safety of her home. Alice decides to fix this problem by modifying Program A into Program B (Figure 2.1). With our tool, she can now compare the two programs to determine whether Program B resolves the issue Program A had. Her visiting mother distrusts Program B and instead proposes Program C (Figure 2.1). Alice can again compare Program B with Program C to see that both programs behave identically.

To help end-users correctly reason about TAP, we designed and evaluated a set of novel user interfaces, powered by formal analysis of TAP programs, that compare TAP programs not just syntactically, but *semantically*. Although our methods can be applied to any arbitrary set of programs, we focus on highly similar variants because the number of differences will be relatively small and therefore tractable to surface to users. While previous work sought to improve TAP understanding by visualizing previous and potential smart home behaviors [53, 16] or explaining certain types of programming errors [18, 17, 50], our interfaces helps users more broadly understand the effects of changes while modifying programs or while selecting among variants to meet their goals.

Traditional **diff**[1] interfaces like those on GitHub or Google Docs compare program text [66, 73, 3, 40]. Our semantic-diff interfaces differentiate programs based on *situational outcomes, general properties,* or *user selection of desired outcomes.* Furthermore, whereas a traditional text-diff interface would highlight syntax differences between Programs B and C (Figure 2.1), our outcome-diff interface would show that the programs actually behave identically. Because these novel semantic-diff interfaces are driven by our formal analysis approach that leverages the relatively small state space of TAP programs, they are specific to TAP, although the concepts can perhaps be adapted to other constrained applications.

To evaluate the effectiveness of these interfaces, we conducted a 107-participant online experiment. We randomly assigned each participant to an interface—either one of our diff interfaces or a control interface showing just the programs themselves. While the control interface and traditional text-diff interface enabled participants to identify differences in short and straightforward programs, our novel semantic-diff interfaces significantly outperformed those interfaces when participants aimed to identify differences between long and complex programs. Notably, our novel interfaces focused on outcomes helped participants accurately complete a wide variety of tasks, and our interface focused on high-level properties helped participants overlook low-level details when appropriate. While (compared to our controls) our novel interfaces helped a significantly larger fraction of participants correctly identify differences between long, complex programs, the time it took to complete tasks did not differ significantly across interfaces.

This work is presented as follows. Section 2.2 defines our terminology and assumptions. We motivate and describe the interaction design of our user interfaces in Section 2.3, and we then describe our formal model of TAP systems and novel algorithms underpinning and

1. Many tools for comparing differences use the name "diff," by analogy to the Unix `diff` utility [31]. We call any comparison interface a "diff" interface.

enabling these interfaces in Section 2.4. We present the methodology of our user study in Section 2.5 and the results in Section 2.6. In Section 2.7, we discuss implications and deployment considerations for TAP systems.

2.2 Terminology and Definitions

To facilitate our presentation of this work, we define the following terms and note the following assumptions. As mentioned in Section 2.1, we term a set of TAP rules a **program**. We term the related programs being compared with our interface **variants**.

A **factor** is any element relevant to the smart home system, including smart devices, the users, and the environment (e.g., weather, time of day). An **attribute** is an aspect of a factor. Example attributes of a user are whether they are asleep and whether they are at home. We will refer to the pair of a factor and an attribute as a **variable** (e.g., "whether the front door is locked" and "whether Alice is asleep"). A **state** is a statement that is true about the variable over some period of time (e.g., "Alice is asleep"), while an **event** is an instantaneous change in the variable's state (e.g., "Alice falls asleep"). A **smart home system state** is the set of all variable states in an environment at that point in time.

In its simplest form, a TAP rule takes the form "IF [trigger] THEN [action]." The **action** is an event that can be automated, such as locking a smart door lock. The trigger activates the action. Out of several trigger types, we use event-state triggers in this work because prior work [30, 9, 68] found this type to be most intuitive for TAP. Event-state triggers consist of an event and zero or more states that must all be true to trigger the action. Note that we refer to just the event part of an event-state trigger as the **trigger** and the states (if any) as the set of **conditions**, such that a rule may have the form "IF [trigger] WHILE [conditions] THEN [action]." We refer to the trigger, the conditions, and the action as rule **subparts**.

For rules whose action reverses the trigger (e.g., "IF the front door unlocks THEN lock the front door," as in Figure 2.1), we assume the rule *prevents* the trigger from occurring, rather than allowing the trigger to proceed and then immediately reversing it. We made this decision to minimize confusion about whether an action occurs.

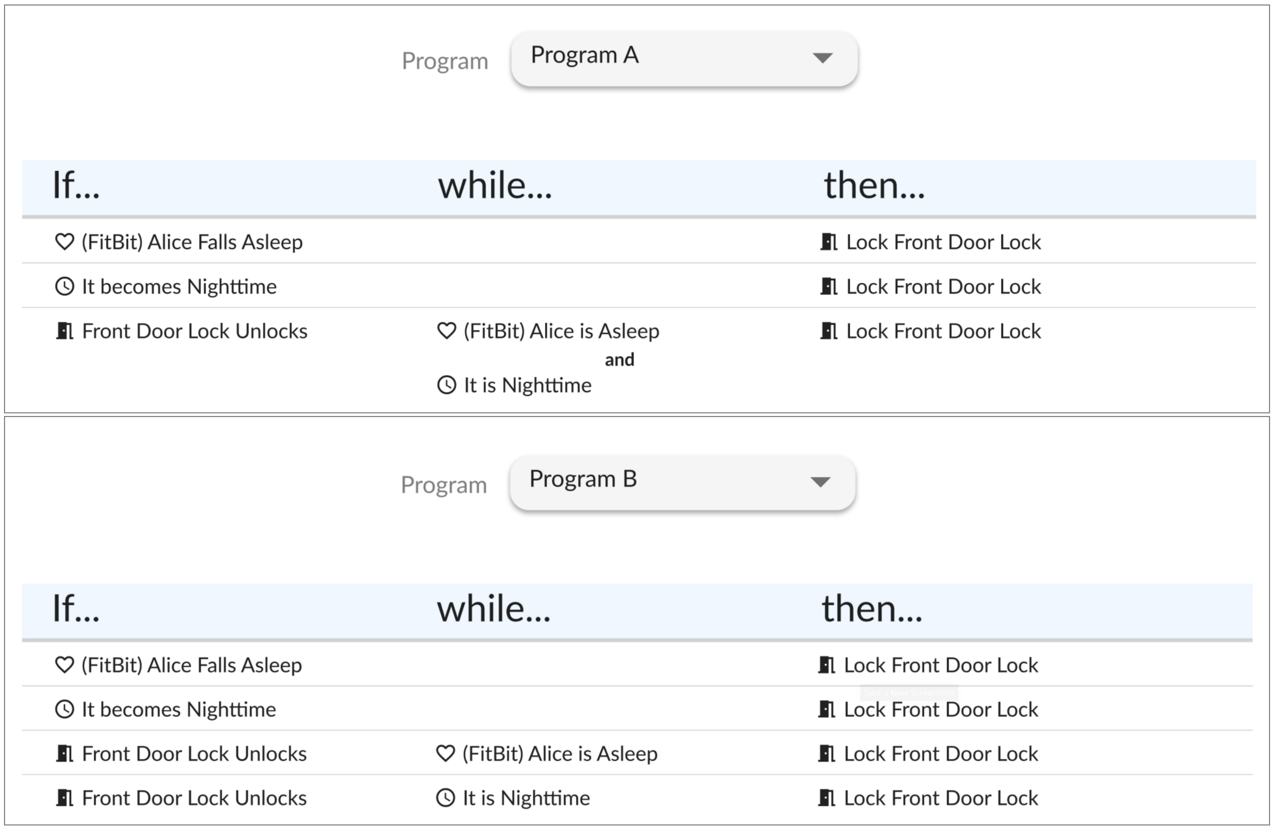
2.3 Diff-Based User Interfaces

In this section, we present our interfaces and describe their goals and interaction design. Section 2.4 complements this section by describing the algorithms underpinning these interfaces. We first discuss our control conditions, *Rules* and *Text-Diff*. The former shows just the programs themselves, while the latter highlights syntax differences. We then describe our semantic-diff interfaces that emphasize TAP differences beyond syntax. We present two interfaces that visualize differences in behavior outcomes between variants (*Outcome-Diff: Flowcharts* and *Outcome-Diff: Questions*) and one that compares abstract properties guaranteed by variants (*Property-Diff*). For each interface, we show how it compares Programs A and B from our running example (Figure 2.1).

2.3.1 Control Conditions

The *Rules* interface displays each program individually (Figure 2.2). Existing systems show trigger-action programs to users individually, so this approach serves as a control condition against diff interfaces. We expected this traditional interface to be sufficient for short, simple variants. A dropdown menu at the top allows users to toggle which program they see. Each row lists a rule with its subparts (trigger, conditions, and action) separated into columns.

The *Text-Diff* interface displays programs side-by-side. It highlights rules that differ, as in traditional diff interfaces (Figure 2.3). We expected *Text-Diff* to help contrast TAP variants that have a small number of key differences. For example, if the differences are simple and independent from the other rules (e.g., if the differences concern the TV and lights, while all other rules relate to the front door lock and whether Alice is



If...	while...	then...
(FitBit) Alice Falls Asleep		Lock Front Door Lock
It becomes Nighttime		Lock Front Door Lock
Front Door Lock Unlocks	(FitBit) Alice is Asleep **and** It is Nighttime	Lock Front Door Lock

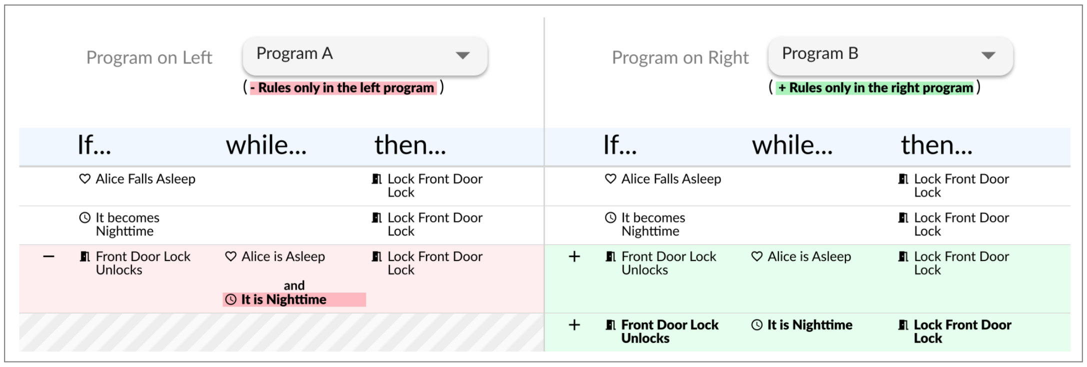
If...	while...	then...
(FitBit) Alice Falls Asleep		Lock Front Door Lock
It becomes Nighttime		Lock Front Door Lock
Front Door Lock Unlocks	(FitBit) Alice is Asleep	Lock Front Door Lock
Front Door Lock Unlocks	It is Nighttime	Lock Front Door Lock

Figure 2.2: *Rules* of Program A (left) and Program B (right) from Figure 2.1. Dropdown menus toggle the program shown.

asleep), we expected *Text-Diff* to be sufficient. Unfortunately, for realistic and practical TAP applications, the text of programs can differ substantially.

We chose *Text-Diff* as a control condition to represent existing diff tools. We piloted several implementations representative of popular tools like GitHub's "split diff" and "unified diff" [66], as well as Google Docs version history changes [73]. Pilot participants preferred the "split diff" design, so we adopted it as our final design.

Like GitHub's "split diff," our *Text-Diff* describes how one can modify the first variant to become the second. Rules unique to the first variant are "deleted" (shown with a minus sign and red background). Rules unique to the second variant are "added" (with a plus sign and green background). The plus and minus signs mitigate the potential inaccessibility of a red/green color scheme to colorblind users. We also bold the text differences. A rule in the



Program on Left: Program A (- Rules only in the left program)

Program on Right: Program B (+ Rules only in the right program)

	If...	while...	then...		If...	while...	then...
	Alice Falls Asleep		Lock Front Door Lock		Alice Falls Asleep		Lock Front Door Lock
	It becomes Nighttime		Lock Front Door Lock		It becomes Nighttime		Lock Front Door Lock
−	Front Door Lock Unlocks	Alice is Asleep and **It is Nighttime**	Lock Front Door Lock	+	Front Door Lock Unlocks	Alice is Asleep	Lock Front Door Lock
				+	**Front Door Lock Unlocks**	**It is Nighttime**	**Lock Front Door Lock**

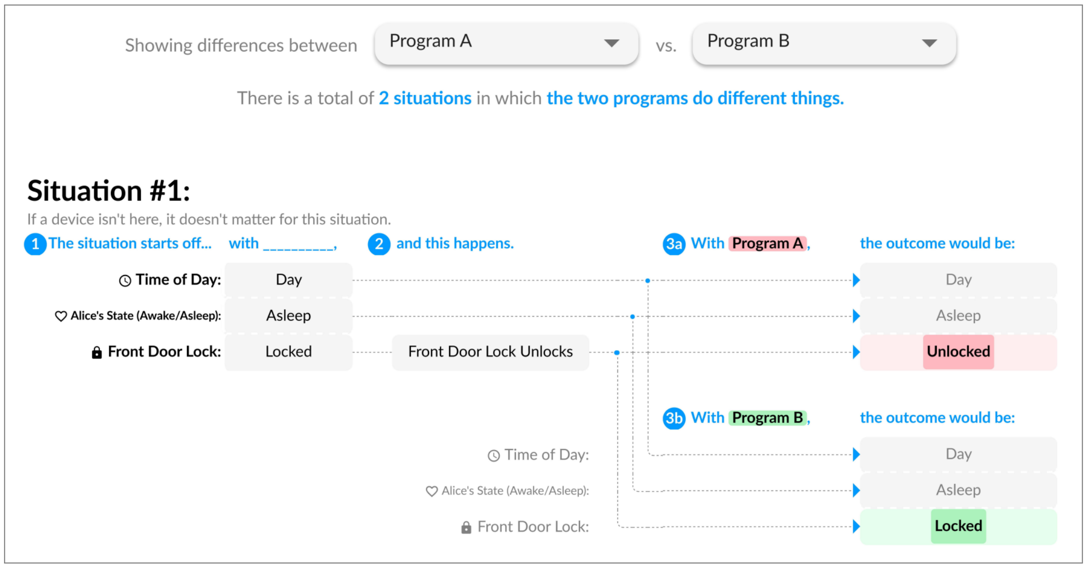
Figure 2.3: *Text-Diff* comparing Programs A and B from Figure 2.1. Similar to GitHub's "split diff" [66], rules unique to a variant have a color background and are preceded by a symbol (red and "-" on the left, green and "+" on the right). When a pair of rules from the two variants is similar, they are aligned on the same row with differences highlighted in darker red or green.

first variant is instead "modified" into a rule in the second if they are similar. We align such rules on the same row and show precise differences with a darker red/green background. In Figure 2.3, the third rule on the left is "modified" to become the third rule on the right by removing "AND it is nighttime." Another rule, "IF front door unlocks WHILE it is nighttime THEN lock the front door," is "added" as the fourth rule on the right. When there are more than two variants, dropdown menus let users select which two to compare.

We redesigned parts of GitHub's "split diff" approach to suit TAP. For a rule to be considered "modified," we defined that they must differ in exactly one subpart. To reduce confusion from misaligned rules, we also sorted the rules of the second variant relative to the first. Unlike traditional software or text documents, trigger-action programs are sets of rules in which the order is flexible. If two variants have identical rules in different orders, a traditional code-diff interface would thus highlight them as different.

2.3.2 Outcome Differences

A shortcoming of traditional diff interfaces is that they do not help users directly discern when a system would take specific actions, nor how the variants cause these actions to differ.


Showing differences between Program A vs. Program B

There is a total of 2 situations in which the two programs do different things.

Situation #1:

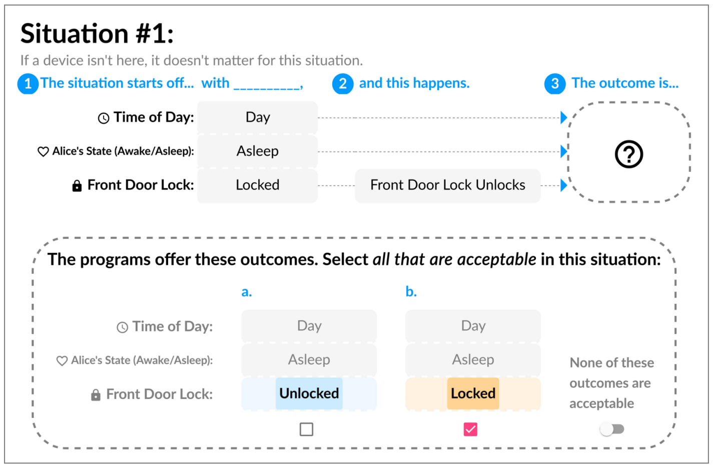
If a device isn't here, it doesn't matter for this situation.

1 The situation starts off... with ________,
2 and this happens.
3a With Program A, the outcome would be:
3b With Program B, the outcome would be:
Time of Day: Day
Alice's State (Awake/Asleep): Asleep
Front Door Lock: Locked
Front Door Lock Unlocks
Day
Asleep
Unlocked
Time of Day:
Alice's State (Awake/Asleep):
Front Door Lock:
Day
Asleep
Locked

Figure 2.4: *Outcome-Diff: Flowcharts* showing a situation in which Programs A and B from Figure 2.1 produce different outcomes.

By seeing or comparing just the text of Programs A and B from Figure 2.1, a user might fail to recognize that when someone attempts to unlock the front door while Alice is asleep during the day, the smart home will prevent them from doing so with Program B, but not A. It is critical to identify the situations in which the variants differ in behavior, but examining only the rules themselves can obscure this information. Here, the *situation* is the context of Alice being asleep while it is daytime and someone is trying to unlock the front door. The *outcome* of Program A is that the front door is unlocked, while with Program B it is locked. Variants produce different outcomes when they take different actions under identical situations.

To help users reason about program differences when they have in mind desired outcomes under specific situations, we present two interfaces that visualize outcome differences. *Outcome-Diff: Flowcharts* uses flowcharts to display all situations in which outcomes differ. *Outcome-Diff: Questions* asks the user to select their desired outcome(s) for these situations through checkboxes, and then summarizes how many, and which, selected outcomes occur under each program variant. To avoid overloading the user, neither interface shows situations



Situation #1:

If a device isn't here, it doesn't matter for this situation.

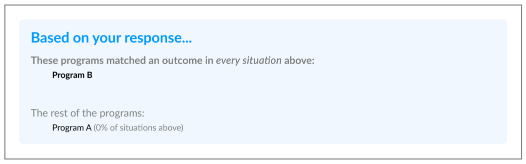
1 The situation starts off... with ________, 2 and this happens. 3 The outcome is...

Time of Day:	Day		?
Alice's State (Awake/Asleep):	Asleep		
Front Door Lock:	Locked	Front Door Lock Unlocks	

The programs offer these outcomes. Select *all that are acceptable* in this situation:

	a.	b.	
Time of Day:	Day	Day	
Alice's State (Awake/Asleep):	Asleep	Asleep	None of these outcomes are acceptable
Front Door Lock:	Unlocked	Locked	
	☐	☑	

(a) The first situation *Outcome-Diff: Questions* shows with Programs A and B from Figure 2.1 as the variants. Users select zero or more desired outcomes.



Based on your response...

These programs matched an outcome in *every situation* above:

Program B

The rest of the programs:

Program A (0% of situations above)

(b) The results of *Outcome-Diff: Questions* appears when the user has made a choice for every situation. Note that for Programs A and B, there are two such situations. These example results occur when the user chooses to have the front door locked in both the first situation (Figure 2.5a) and the second situation.

Figure 2.5: Snippet of *Outcome-Diff: Questions* with Programs A and B from Figure 2.1 as variants.

with identical outcomes.

***Outcome-Diff: Flowcharts* shows all situations in which two variants produce different outcomes** (Figure 2.4). Via dropdown menus, a user chooses two variants to compare and sees a series of flowcharts highlighting differences in outcomes. The interface also states the number of situations in which outcomes differ between those variants. Each flowchart shows a situation. For Programs A and B, *Outcome-Diff: Flowcharts* shows two situations in which the two programs produce different outcomes. The first situation (Figure 2.4) starts with Alice asleep during the day while the front door is locked. Then, a hypothetical event—the front door unlocking—occurs. Program A results in the front door being *unlocked*, while Program B results in it being *locked*. Variable state differences use the same color scheme as text differences in *Text-Diff*, and we took the same steps to enhance accessibility.

***Outcome-Diff: Questions* instead asks the user to indicate what outcomes, if any, are desirable in particular situations**, as shown in Figure 2.5. It shows the same situations as *Outcome-Diff: Flowcharts*, albeit in this revised question format. Whereas

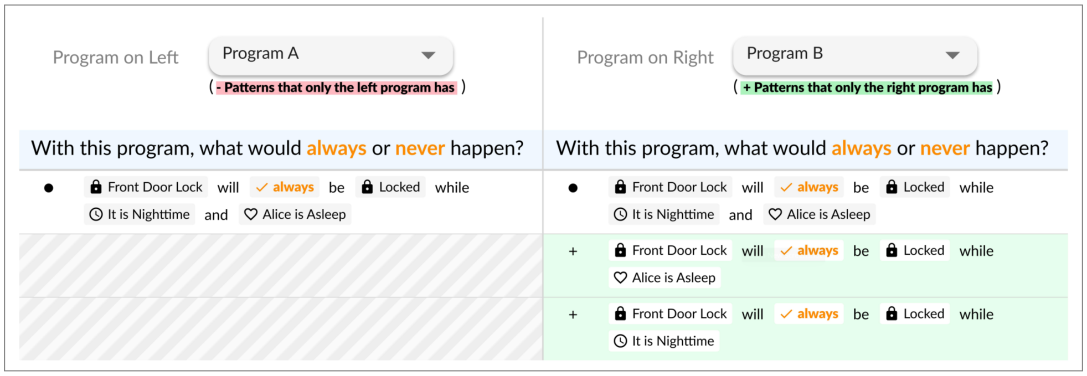
Outcome-Diff: Flowcharts, *Text-Diff*, and *Property-Diff* all compare exactly two variants at a time, pairwise comparisons are tedious and overwhelming if there are many variants to compare. The approach of *Outcome-Diff: Questions* enables the user to quickly identify variants with the desired behaviors among a large set of variants. For each specific situation, the interface lists all of the different outcomes caused by at least one of the variants under that situation. The user selects their desired choice(s), or specifies that they have no preference. Figure 2.5a shows an example of the same situation in Figure 2.4 for Programs A and B. Below the situation are the outcome choices. Differences between outcomes have orange backgrounds in selected outcomes and blue backgrounds in unselected outcomes.

For each program variant, *Outcome-Diff: Questions* tracks the number of situations in which the user-selected outcome matches what would occur in that variant. Once the user has responded to all of the given situations, *Outcome-Diff: Questions* presents the percentage of situations for which each variant would have an acceptable outcome. A variant that satisfies more situations is more likely to match the user's intent. In the case of Figure 2.5, the interface shows that Program B matches a selected outcome in all situations where outcomes differ, while Program A matches none of them.

2.3.3 Property Differences

Sometimes, the user might care mainly about general trends or guarantees, such as whether the door will always be locked when Alice is asleep, as opposed to specific behaviors in specific situations. It is difficult for users to extract high-level trends from any interface presented so far, especially when variants have many differences.

***Property-Diff* helps users reason about broader behaviors in their automated systems by contrasting high-level properties held by the two variants** (Figure 2.6). In particular, our interface highlights *safety properties*, which are informally defined as statements indicating that "nothing bad happens in the system's execution" [5]. For instance, all


Program on Left
Program A
(- Patterns that only the left program has)
Program on Right
Program B
(+ Patterns that only the right program has)

With this program, what would always or never happen?	With this program, what would always or never happen?
● Front Door Lock will ✓ always be Locked while It is Nighttime and Alice is Asleep	● Front Door Lock will ✓ always be Locked while It is Nighttime and Alice is Asleep
▨	+ Front Door Lock will ✓ always be Locked while Alice is Asleep
▨	+ Front Door Lock will ✓ always be Locked while It is Nighttime

Figure 2.6: *Property-Diff* comparing the safety properties of Programs A and B from Figure 2.1.

Table 2.1: Properties that *Property-Diff* supports. The terms [*device_state*] and [*device_event*] refer to a device variable's state or event, while [*external_state*] refers to the state of a variable that cannot be controlled, like attributes of users and environmental factors. A [*state*] can be the state of any variable.

Property Template Based on AutoTap [94]	**Example Phrasing in the *Property-Diff* Interface**
[*device_state*] will [*always/never*] be active.	[*The lights*] will [*always*] be [*on*].
[*device_state*] will [*always/never*] be active while [*external_state*$_1, \cdots,$ *external_state*$_n$].	[*The lights*] will [*always*] be [*on*] when [*it is nighttime*] and [*Alice is awake*].
[*state*$_1, \cdots,$ *state*$_n$] will [*never*] be active together.	The smart home will [*never*] have [*the lights on*] and [*the window curtains open*] at the same time.
[*device_event*] will [*only/never*] happen when [*state*$_1$, ..., *state*$_n$].	[*The lights*] will [*only*] [*turn on*] when [*it is nighttime*].

three programs in Figure 2 have the safety property that "the front door will always be locked when Alice is asleep and it is nighttime." We designed this interface to be useful when differences in low-level rules or outcomes are unimportant and may even be burdensome to users.

The properties *Property-Diff* presents are based on templates from Zhang et al. [94] and are listed in Table 2.1. We excluded templates they found to confuse users, such as "[*state*$_1$, ..., *state*$_n$] will [*always*] be active together." We also condensed equivalent templates and separated those whose meaning can differ based on the number of device variables. Finally, we excluded timing-related properties, such as "[*state*] will [*never*] be active for more than [*duration*]," because they require different transition-system analyses than properties unrelated to timing (see Section 2.4 and [94]).

To maximize consistency across interfaces, we followed the layout and visual elements of *Text-Diff* for contrasting properties. Pilot participants did not understand the phrase “safety property,” so we listed them as “patterns that are always or never true about the smart home.” In addition to the properties that differ, *Property-Diff* also shows properties held by both variants. Pilot study participants preferred seeing all properties to gauge more comprehensively whether any variant satisfied the user study tasks. Unlike *Text-Diff*, we did not define how two different properties can be similar. Doing so requires more careful consideration of what makes two properties “similar,” which we leave to future work.

2.4 Algorithms For Computing Diffs

In this section, we present the algorithms underpinning the interfaces that we presented in Section 2.3. Section 2.4.1 shows how we compare text differences across variants for *Text-Diff.* Section 2.4.2 introduces transitions systems, which we use to represent TAP variants in a home. They serve as the basis for our algorithms for semantic-diff interfaces. Section 2.4.3 explains our algorithms to identify outcome differences across variants for *Outcome-Diff: Flowcharts* and *Outcome-Diff: Questions*. Section 2.4.4 introduces how we generate property differences across variants for *Property-Diff.*

2.4.1 Text-Diff

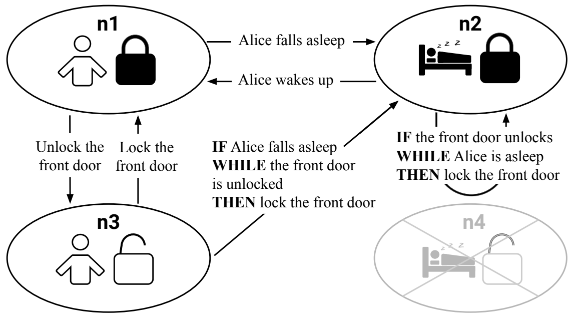
The main task in populating the *Text-Diff* interface algorithmically is to identify potential “modified” rules, which are pairs of rules from the two variants that are most similar to each other. The set of candidates is the Cartesian product of the two sets of rules from the two variants. Per our definition of “modified” rules in Section 2.3, we eliminate all candidates in which the rules differ by more than one subpart. We calculate the differences within each remaining pair of rules to be the number of conditions in which the pair differs, or 1 if the pair differs in their triggers or their actions (never both). We then choose the largest unique set of rule pairs with the smallest differences to be the set of "modified" rules.

2.4.2 Modeling the Home as a Transition System

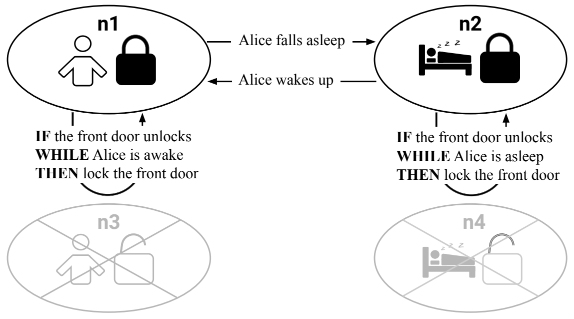
All of our semantic-diff interfaces require reasoning about how particular TAP variants differ in behavior. To this end, we model each TAP variant and the smart home as a transition system [5, 94]. Each transition system incorporates rules of the variant as well as the subset of the home's devices (and their possible states) and environmental factors (e.g., weather, light levels, time of day) that affect or are triggered by these rules.

A node in the transition system represents every device and every environmental factor from the variant being in a particular state. Recall that each device or environmental factor in the home is represented by a variable. The set of states (nodes) in a given transition system is thus the Cartesian product of the set of states for each variable. For example, a system with n binary variables would have 2^n states. While a home with many devices and relevant environmental factors would seemingly require a huge transition system to model, the transition system actually only needs to include the devices and environmental factors directly related to the rules in the TAP variants being compared. For instance, if no rule in the TAP variants being compared is triggered by, nor acts upon, the home's thermostat, the thermostat can be excluded entirely from the transition system. Furthermore, a variable with a large number of possible states (e.g., the home's temperature) can be discretized. For instance, if the only aspect of temperature relevant to any rule in the variants being compared is whether or not the home is under 65°F, temperature can be treated as a binary variable. As a result, even for a home with many devices, the transition system typically remains small. For our user study (Section 2.5), the largest system—Task 6 (Abstraction)—had 7 binary variables and thus 128 states (nodes).

Transitions (edges) in the transition system reflect events that change one or more variables, which intuitively means that one or more devices or environmental factors has changed


n1
n2
n3
n4
Alice falls asleep
Alice wakes up
Unlock the front door
Lock the front door
IF Alice falls asleep **WHILE** the front door is unlocked **THEN** lock the front door
IF the front door unlocks **WHILE** Alice is asleep **THEN** lock the front door

(a) TS_1, a transition system for a trigger-action program consisting of two rules: "IF Alice falls asleep WHILE the front door is unlocked THEN lock the front door" and "IF the front door unlocks WHILE Alice is asleep THEN lock the front door." The first rule redirects transition $n3 \rightarrow n4$ to $n3 \rightarrow n2$. The second rule redirects the transition $n2 \rightarrow n4$ to $n2 \rightarrow n2$ (a self-transition). Assuming $n1$ is the initial state, this program ensures that $n4$ is unreachable.


n1
n2
n3
n4
Alice falls asleep
Alice wakes up
IF the front door unlocks **WHILE** Alice is awake **THEN** lock the front door
IF the front door unlocks **WHILE** Alice is asleep **THEN** lock the front door

(b) TS_2, a transition system for a trigger-action program consisting of two rules: "IF the front door unlocks WHILE Alice is awake THEN lock the front door" and "IF the front door unlocks WHILE Alice is asleep THEN lock the front door." Assuming $n1$ is the initial state, this program ensures both $n3$ and $n4$ are unreachable.

Figure 2.7: Transition systems (TS_1 and TS_2) of two TAP variants, both focusing on whether Alice is awake and whether the front door is locked. The system states are labeled $n1$ through $n4$. Arrows represent valid transitions. Alice is awake in $n1$ and $n3$, and asleep in $n2$ and $n4$. The front door is locked in $n1$ and $n2$, and unlocked in $n3$ and $n4$. If a rule triggers (redirects) a transition, we label the corresponding transition arrow with the rule. Unreachable nodes are colored gray and crossed out.

its state. Initially, these edges will represent manual transitions (e.g., a light can change state if someone manually turns it on) and natural environmental transitions (e.g., day will turn to night). When there are no automation rules, every pair of smart home states that differ in the state of exactly *one* variable will typically be connected by two transitions, one from each state to the other. However, this pattern does not hold true if not all transitions for a device are valid. For instance, in modeling a device that must always temporarily be in a "warming up" state before it reaches the "on" state, there will not be an edge from "off" to "on" nodes. As with Zhang et al. [94], we require that the valid transitions for a given type of variable be pre-specified as part of an overall model of a smart home.

Rules redirect transitions in the graph. Specifically, the in-transition of a state representing a particular rule's triggering event and conditions will be redirected to the state reflecting the eventual outcome of the action specified by the rule.

To further illustrate transition systems, we partially model our running example from Section 2.1. The states of this transition system will be the Cartesian product of whether Alice is asleep and whether the front door is locked (Figure 2.7), resulting in four states:

- $(\text{Alice}_{\text{awake}}, \text{door}_{\text{unlocked}})$
- $(\text{Alice}_{\text{awake}}, \text{door}_{\text{locked}})$
- $(\text{Alice}_{\text{asleep}}, \text{door}_{\text{unlocked}})$
- $(\text{Alice}_{\text{asleep}}, \text{door}_{\text{locked}})$

If Alice wakes up, which is a variable state change, the system transitions from $\text{Alice}_{\text{asleep}}$ to $\text{Alice}_{\text{awake}}$. If someone manually locks the door, the system transitions from $\text{door}_{\text{unlocked}}$ to $\text{door}_{\text{locked}}$.

Again, rules redirect these transitions. For instance, if Alice is awake with the door unlocked and then falls asleep, without any rules the system would move from state $n3$ to state $n4$ of Figure 2.7a. However, the rule "IF Alice falls asleep WHILE the front door is unlocked THEN lock the front door" instead redirects this edge to node $n2$ because the rule's action is to lock the front door.

Our analyses for *Outcome-Diff* and *Property-Diff* rely on reachability analysis, using breadth-first search to identify reachable states in the system. This search process requires an initial state from which to start. Inadvertently choosing an invalid (unreachable) state as the starting point would result in the reachability analysis being incorrect, mistaking some unreachable states as reachable and vice versa. Most intuitively, real-world systems can choose the current state of the home as this initial state. In our user study, we choose a state that satisfies the goal of the task as the initial state. For example, if the goal is to make sure the door is always locked while Alice is asleep, we define the initial state to be any state except for the door being unlocked while Alice is asleep.

2.4.3 Outcome-Diffs

To identify all situations in which variants produce different outcomes, we compare their transition systems. Each variant is represented by its own transition system. The key intuition for detecting differences in outcomes is to compare the out-transitions for a given (corresponding) state across variants. If a given (corresponding) transition from this state differs across program variants, in that its destination state does not correspond across variants, that means the outcome of that event is different.

The *Outcome-Diff* interfaces share similar algorithms, so we detail only the *Questions* algorithm for n variants, presenting its pseudocode as Algorithm 1. *Flowcharts* uses this algorithm for $n = 2$.

First, we generate the transition systems $TS_1 \cdots TS_n$ for the n variants. We then find the nodes that are reachable in every transition system, which are the system *states* that are possible with all of the variants. For all outgoing edges of these nodes (i.e., every possible *event* happening from these *states*), we identify the actions they would trigger with $Variant_1 \cdots Variant_n$. For each situation, we group the variants based on the actions they take, combining groups for which the actions lead to identical outcomes. We also merge the situations to minimize redundancy. We then convert the remaining situations to multiple-choice questions, each asking "what should the program do when *event* happens in *state*," with the choices being all possible outcomes from $Variant_1 \cdots Variant_n$.

We present an example for $n = 2$ (identical to *Outcome-Diff: Flowcharts*) with Figure 2.7. The first program consists of two rules: "IF Alice falls asleep WHILE the front door is unlocked THEN lock the front door" and "IF the front door unlocks WHILE Alice is asleep THEN lock the front door." The second program consists of two rules as well: "IF the front door unlocks WHILE Alice is awake THEN lock the front door" and "IF the front door unlocks WHILE Alice is asleep THEN lock the front door." Assuming $n1$ is the initial state, the two respective transition systems share two possible *states*: $n1$ and $n2$. From *state* $n1$, the possible events are Alice falling asleep and the door becoming unlocked (by someone or another rule). In the left transition system, when the front door becomes unlocked while Alice is awake at $n1$, the system allows this transition to $n3$. In the right transition system, the system would instead transition back to $n1$ with Alice awake and the door still locked. For *Outcome-Diff: Questions*, the algorithm would generate this question (in flowchart form): "The situation starts off with Alice awake and the front door locked. Now the front door unlocks. What should the outcome of this situation be?" The choices would be "have Alice awake and the front door unlocked" and "have Alice awake and the front door locked." In *Outcome-Diff: Flowcharts*, a flowchart would show this situation with the two choices as diverging outcomes (Section 2.3.2).

When there are multiple conditions or actions, we combine situations that only differ in conditions that do not affect the outcome. For example, if two situations are identical in conditions and in the set of outcomes, except that Alice is asleep in one situation and awake in the other, we combine them into a single situation and do not show whether Alice is asleep on the interface.

2.4.4 Property-Diff

Our algorithm underlying *Property-Diff* centers on analyzing the unreachable nodes in variants' transition systems. The key intuition is that the states of a given variable or given combination of variables that are always unreachable can be directly mapped to human-intelligible safety properties for the *Property-Diff* interface.

We first extract safety properties from each variant. We then separate the safety properties shared by both variants. The safety properties we consider (Table 2.1) consist of

```
Input  : $Variant_1, \cdots, Variant_n$
Output: $situationDiffs$, a list of ($startState$, $event$, $behaviorMap$) where variants
        behave differently by taking different actions. $behaviorMap$ is a map from
        behaviors to variant ids.
Function findSituationDiffs($Variant_1, \cdots, Variant_n$)
    $TS_1 \cdots TS_n$ := transition systems of smart home implementing
      $Variant_1 \cdots Variant_n$;
    $R_1 \cdots R_n$ := sets of reachable states in $TS_1 \cdots TS_n$;
    $R := R_1 \cap \cdots \cap R_n$;
    $situationDiffs := \emptyset$;
    for Every state in $R$ do
        for Every event that can happen from state do
            $beh_1 \cdots beh_n$ := Actions triggered by $Variant_1 \cdots Variant_n$ when $event$
              happens under $state$;
            if $beh_1, \cdots beh_n$ are not all the same then
                $behaviorMap := \{\}$ (empty map);
                for $i$ in $1 \cdots n$ do
                    if $beh_i$ is not in $behaviorMap$ then
                        $behaviorMap[beh_i] := \emptyset$;
                    $behaviorMap[beh_i] := behaviorMap[beh_i] \cup \{i\}$;
                $situationDiffs := situationDiffs \cup \{(state, event, behaviorMap)\}$;
    return $situationDiffs$
```

Algorithm 1: Pseudocode for the *Outcome-Diff: Questions* algorithm. We identify situations (a system state and an event) that result in any of the n variants producing different outcomes. *Outcome-Diff: Flowcharts* uses this algorithm for $n = 2$.

properties based on states (**S-properties**, such as "the door will never be unlocked when Alice is asleep") and properties based on events (**E-properties**, such as "the door will never *unlock* when Alice is asleep"). Algorithm 2 presents our pseudocode. We find S- and E-properties separately.

Our algorithm considers that safety properties can be stated as logical expressions, and therefore multiple logical expressions are equivalent. For example, the S-property "the front door will always be locked" implies other S-properties, such as "the front door will always be locked *while Alice is awake*" and "the front door will always be locked *while Alice is asleep*." These latter two properties can combine to yield the first by plugging the variable states into a Boolean expression in disjunctive normal form—"*(Front door being locked AND*

Alice being awake) | (Front door being locked AND Alice being asleep)"—and simplifying this expression. This approach has been used in related applications, such as simplifying smart building rulesets [71]. Note that variables with more than two states (e.g., light hue color) and range variables (e.g., temperature) require a few more steps, but can also be simplified in this manner. E-properties can also be written in the form of other E-properties and combined with logic minimization. Assuming that the front door is locked in the initial state, the first S-property can also be stated as the E-property "the front door will never *unlock*" and (implicitly) the E-property "the front door will never *lock*" as it is already locked.

Displaying a large number of properties would likely overwhelm users. Therefore, our algorithm extracts properties in their most simplified form, which in this case is the first S-property. In addition, because positive statements are typically easier to understand than negative ones, when possible we convert properties about states or events that can *never* occur into properties about states or events that will *always* occur. For example, "the front door will never be unlocked" is restated as "the front door will always be locked."

We first extract S-properties by identifying unreachable nodes in each system. An unreachable node indicates a corresponding state that can never hold true in that system. For example, if the node corresponding to $(\text{Alice}_{\text{asleep}}, \text{door}_{\text{unlocked}})$ is unreachable, then the property "the front door will never be unlocked while Alice is asleep" is true for the system. Instead of converting each node into a property in this manner, we perform logic minimization over the set of such nodes to merge them based on common variable states, and then generate corresponding S-properties. For example, if the node corresponding to $(\text{Alice}_{\text{awake}}, \text{door}_{\text{unlocked}})$ is unreachable in addition to the node we just mentioned, then we merge these two nodes to be $(\text{door}_{\text{unlocked}})$, yielding the property "the front door will never be unlocked." This approach lets us generate one general property, as opposed to two specific properties. Our implementation uses the SymPy library [54] for minimization, which relies on the Quine-McCluskey algorithm [52]. Other logic minimization algorithms would be valid as well. We then identify nodes unreachable in both systems and repeat the same process on them to identify S-properties shared by both systems. This step not only helps with the next step, in which we determine S-properties unique to one variant but not the other, but it also outputs the shared S-properties that our interface shows in addition to unique S-properties. Finally, we subtract the shared S-properties from the S-properties of each transition system. Each system's remaining S-properties are unique to that system, and thus that variant.

As an example, we run our analysis on TS_1 and TS_2 in Figure 2.7. With $n1$ as the initial state, we find that $n4$ $(\text{Alice}_{\text{asleep}}, \text{door}_{\text{unlocked}})$ is unreachable in both TS_1 and TS_2. Therefore, the two systems both have the property "the front door will never be unlocked while Alice is asleep." This is also the only S-property held by TS_1. We find that both $n3$ $(\text{Alice}_{\text{awake}}, \text{door}_{\text{unlocked}})$ and $n4$ are unreachable in TS_2, which produces the property "the front door will always be locked." The shared S-property cannot be subtracted from this property. Therefore, we will tell the user that this property is unique to TS_2. Alternatively we could say that the property unique to TS_2 is "the front door will always be locked *while Alice is awake*," but we designed the algorithm to quickly deduce properties at the most abstract level and avoid outputting too many properties when a few would be equivalent or imply them.

For E-properties, we identify self-transitions and repeat the same algorithm as for S-properties. A self-transition indicates that an event can never occur, possibly under some situation (e.g., "the front door will never unlock when Alice is awake" based on the self-transition from $n1$ in TS_2, but not TS_1). Some E-properties are implied by S-properties because their corresponding self-transitions contribute to an unreachable node. Therefore, we ignore these self-transitions. This is the case for all self-transitions in TS_1 and TS_2 of Figure 2.7. Therefore, we do not show any E-properties for them.

```
Input  : $TS_1$, $TS_2$: transition systems of smart home implementing $Variant_1$,
         $Variant_2$
Output: $allProperties$: 3-tuple of properties unique to $Variant_1$, properties unique
         to $Variant_2$, and properties held by both variants
Function findAllProperties($TS_1$, $TS_2$)
    $U_1$ := set of nodes unreachable in $TS_1$;
    $U_2$ := set of nodes unreachable in $TS_2$;
    $U_{\text{both}} := U_1 \cap U_2$;
    // $S_{\text{both}}$: S-properties shared by both variants
    $S_{\text{both}}$ := toSproperties(minimizeLogic($U_{\text{both}}$));
    // $S_1, S_2$: unique S-properties of each variant
    $S_1$ := toSproperties(minimizeLogic($U_1$)) $-S_{\text{both}}$;
    $S_2$ := toSproperties(minimizeLogic($U_2$)) $-S_{\text{both}}$;

    $T_1$ := set of self-transitions from reachable nodes that do not contribute to $U_1$ in
     $TS_1$;
    $T_2$ := set of self-transitions from reachable nodes that do not contribute to $U_2$ in
     $TS_2$;
    $T_{\text{both}} := T_1 \cap T_2$;
    // $E_{\text{both}}$: E-properties shared by both variants
    $E_{\text{both}}$ := toEproperties(minimizeLogic($T_{\text{both}}$));
    // $E_1, E_2$: unique E-properties of each variant
    $E_1$ := toEproperties(minimizeLogic($T_1$)) $-T_{\text{both}}$;
    $E_2$ := toEproperties(minimizeLogic($T_2$)) $-T_{\text{both}}$;

    $uniqueProperties_1 := S_1 \cup E_1$;
    $uniqueProperties_2 := S_2 \cup E_2$;
    $properties_{\text{both}} := S_{\text{both}} \cup E_{\text{both}}$;
    return $(uniqueProperties_1, uniqueProperties_2, properties_{\text{both}})$
```

Algorithm 2: Pseudocode for the *Property-Diff* algorithm. We identify properties that are unique to each variant, as well as properties shared by both variants. "minimizeLogic()" plugs nodes or edges into a logic expression in disjunctive normal form, simplifying this expression. Self-transitions that contribute to unreachable nodes are those that, due to their redirection, helped make the original destination node unreachable.

2.4.5 Scalability

Although real-world systems may have large programs and transition systems, we expect users will want to compare subsets of rules related to a specific task (e.g., controlling the door lock) to related sets of rules, as described in Section 2.1. While our algorithms could become computationally intractable for a sufficiently large transition system, we believe transition systems for realistic tasks will generally be small from designing the study tasks. For example, in a realistic TAP diff task—Task 6 (Abstraction) with 7 binary attributes—it took only 8 seconds on a commodity laptop to generate results from scratch (showing 19 situations) for *Outcome-Diff: Flowcharts*, and 5 seconds for *Property-Diff*. Precomputation and caching would further reduce the time required to run the algorithms and help the approach scale further.

2.5 User Study Methodology

To evaluate whether our interfaces could help users understand TAP variants, we performed a 107-participant online user study. We studied the following research questions:

- **RQ1:** Compared to *Rules* or *Text-Diff*, do semantic-diff interfaces equip users to more accurately choose the correct program variant(s) out of a set of prospective variants to match a motivating goal?
- **RQ2:** How does the relative performance of these interfaces compare across programs with different characteristics (e.g., length, complexity, number of prospective variants)?
- **RQ3:** Do specific interfaces help users reason about TAP program differences more (a) confidently and (b) quickly?

To address these questions, we designed six program-comparison tasks modeled after potential scenarios in which the user encounters variants (Section 2.5.3). We implemented the interfaces for participants to use. Each participant would complete these tasks in a randomized order using a randomly assigned interface. For each task, participants could choose to see the programs themselves (listed in prose) by clicking on a "program button." Our tutorial encouraged participants to focus on the interface, briefly mentioning that this "program button" feature was available. As described in Section 2.3, we piloted interface designs on users with various technical backgrounds before the study to refine our designs.

2.5.1 Recruitment

We recruited participants with Prolific Academic [67], a recommended alternative [70] to other recruitment platforms like Amazon Mechanical Turk. Participants had to be 18 years or older with US nationality to take part in the study. We also required them to have a 90% or higher approval rate from at least 10 previous submissions. As we expected our interfaces to be particularly helpful for non-technical users, our recruitment description emphasized that we did *not* require experience with programming or smart homes. We asked participants to take the study on a computer, not a phone.

2.5.2 Study Overview

We first eased the participants into TAP and their randomly assigned interface. After consenting to the study, participants read a one-page description of trigger-action programs. They then completed an interactive tutorial of their assigned interface on a sample task. If a participant failed two of the three simple attention check items, the study terminated early. Otherwise, participants continued onto the main portion of the study for their interface.

In the main portion, participants completed the six program-comparison tasks in a randomized order. For each task, we recorded participants' responses and durations. After each task, we asked participants to rate on a 7-point Likert scale how much they agreed with the following three statements: "I am confident that my answer is correct"; "I found the task mentally demanding"; and "I found the interface helpful in completing the task." We also asked them to briefly describe their approach to completing the task and how the interface helped or hindered.

Following the six tasks, we collected more in-depth information about participants' experiences using the interfaces, as well as relevant background that could have influenced their performance and experience. We asked participants to describe their general approach more specifically, including the parts of the interface that were most helpful, least helpful, and most confusing. To illustrate the level of detail we were looking for in their descriptions, we provided an example describing a hypothetical approach for accepting Facebook friend requests. We also asked them whether they relied mostly on the interface itself, the programs, or both. We quantitatively gauged interface usability with the 7-point version of the System Usability Scale (SUS). The study concluded with questions about the participant's demographics and technology background.

This study was approved by the UChicago IRB. We compensated each participant \$10.00. The average completion time was slightly over an hour. We include the full survey instrument and the text of the tasks in our online supplementary materials [96].

2.5.3 Task Design

To understand how our interfaces can help users, we designed the tasks to emulate scenarios in which users need to compare variants (Table 2.2 and Table 2.3). We included at least one task for each interface, including the controls, that we expected to highlight the unique advantages of the interface. Each task asked the participant to compare between two or more programs. In all tasks, we evaluated whether participants could identify differences across variants that the task requested. Our tasks followed these hypothetical scenarios:

1. Task 1 (Straightforward): Given an original program, a desired behavior extension, and modified versions of this program, could the participant determine which of the modified versions maintained the original program behaviors, but extended them as specified?

2. Task 2 (Simple Logic) and Task 6 (Abstraction): Given an original program, some observations about its undesirable behaviors, and modified versions of this program, could

Table 2.2: Overview of the complexities of the tasks.

Task	Complex Programs	Situations with Different Outcomes	Differ in Properties	Many Variants
1 - Straightforward		✓		
2 - Simple Logic		✓		
3 - Redundant Programs	✓	✓	✓	
4 - Hidden Similarity	✓	✓	✓	
5 - 27 Variants		✓	✓	✓
6 - Abstraction	✓	✓	✓	

the participant determine which of the modified versions would exhibit the desirable behaviors instead?

3. Task 3 (Redundant Programs) and Task 4 (Hidden Similarity): Given an original program, its goal, and modified versions of this program, could the participant determine which of the modified versions met the same goal as the original program?

4. Task 5 (27 Variants): Given multiple programs and a set of goals, could the participant determine which of the programs meet all of the goals?

We designed the tasks with the following characteristics to best highlight each interface. In general, we expected *Outcome-Diff* interfaces to outperform the controls on tasks with complex programs. We defined complex programs to be long programs with many rules affecting the same variables, which may hinder participants from tracking all behavior differences. We arbitrarily decided that a long program would contain at least six rules, with some rules containing three or more conditions. Meanwhile, we expected participants using the control interfaces to do well on tasks with variants that were not complex. Therefore, we designed Tasks 3 (Redundant Programs), 4 (Hidden Similarity), and 6 (Abstraction) to have complex programs, and Tasks 1 (Straightforward) and 2 (Simple Logic) to have simple programs. Variants in the former three tasks also differed in properties, for which *Property-Diff*

Table 2.3: Detailed summary of the tasks.

1 - Straightforward	Given an original program and a modified version, the participant decides whether the modified version does exactly what the original program does, except it also turns off the faucet when Alice leaves the bathroom.
2 - Simple Logic	Given an original program and two modified versions, the participant decides which of the two versions (if any) would turn off the AC when neither Alice nor Bobbie is at home, but does not affect the AC if either one of them is home.
3 - Redundant Programs	Given an original program that meets a specified goal and a modified version, the participant decides whether the modified version also meets the specified goal. The goal is to secure the home by keeping the security camera on when the front door is unlocked or Alice is asleep. Both programs contain redundant rules.
4 - Hidden Similarity	Given an original program that meets a specified goal and two modified versions, the participant decides which of the two versions (if any) would meet the specified goal. The goal is to keep exactly one of three windows open in the house at any given time.
5 - 27 Variants	Given 27 programs that meet the same goal, the participant decides which of the programs (if any) would meet a second goal. The first goal is to ensure that the living room window being open, the TV being on, and the Roomba being on will never occur simultaneously. The second goal is to ensure that Alice gets to enjoy fresh air from the living room window without disruption to her TV time.
6 - Abstraction	Given an original program that fails to meet a specified goal and two modified versions, the participant decides which of the two versions (if any) can meet the goal. The goal is to save electricity by ensuring that only one of eight smart devices (AC, coffee pot, lights, TV, speakers, and the Roomba) is on at any given time.

should help. To see whether participants would benefit from the form-based interaction of *Outcome-Diff: Questions*, we designed Task 5 (27 Variants) to have many variants. In all tasks, variants had situations with different outcomes, for which we expected *Outcome-Diff* participants to do well even if the variants were not long enough to be complex.

2.5.4 Statistical Analyses

Our statistical analysis approach depended on whether our variables of interest were categorical or numeric. For both cases, we first conducted an omnibus test to determine whether the variable of interest varied by interface. For significant omnibus results (using $\alpha = 0.05$), we then conducted pairwise comparisons across all interfaces to determine which interfaces varied significantly from which other interfaces. We were particularly interested in how our *Outcome-Diff* and *Property-Diff* interfaces compared to the two control conditions, which reflected existing interfaces. Thus, we report the results of these comparisons most prominently. To minimize the possibility of Type I errors due to multiple testing, we corrected p-values within each family of omnibus tests and within each set of pairwise comparisons using the Holm method.

For both omnibus and pairwise comparisons between interfaces for categorical data, we used Fisher's Exact Test. We chose Fisher's Exact Test instead of the more familiar Pearson's χ^2 test because the latter is considered unreliable when expected cell counts are smaller than 5, which was often the case in our data. These categorical comparisons analyzed how the correctness of participants' answers differed across interfaces for each of the six tasks, as well as how the use of the program button varied across interfaces for each of the six tasks. Thus, each instance of Fisher's Exact Test was run on a 2×5 contingency table (binary outcomes across five interfaces).

For omnibus comparisons of numeric data, we used the Kruskal-Wallis H test. For pairwise comparisons, we used its analogue for 2 groups, the unpaired Wilcoxon rank-sum test, also known as the Mann-Whitney U test. These non-parametric tests are appropriate for data that is not normally distributed and for ordinal data, which was the case for our quantitative data. In particular, we analyzed how the number of tasks participants answered correctly, the number of tasks for which participants used the program button, System Usability Scale scores, and total time taken (summed across tasks) varied by interface. We also tested whether participants' Likert-scale responses to the following three prompts (averaged across tasks) varied by interface: their confidence in their answers to the tasks, whether they considered the tasks demanding, and whether they considered the interface helpful in completing the tasks.

We also created regression models to understand how both the assigned interface and numerous demographic characteristics correlated with the number of tasks each participant answered correctly and the number of tasks for which they used the program button. Because both factors are counts, we created Poisson regression models. The independent variables (IVs) were the assigned interface, number of tutorial questions answered correctly, and the participant's age, gender, education level, computer science background, familiarity with

IFTTT, and ownership of IoT devices. For categorical variables with many categories, we binned similar categories. Using the same IVs, we also created generalized linear models for the total time a participant took across the six tasks and their average confidence rating across tasks, both of which we treated as continuous. For all regressions, we created (and report) a parsimonious model developed through backward selection by Akaike Information Criterion (AIC). Our supplementary materials [96] provides the full regression tables. For brevity, we report only p-values here.

2.5.5 Limitations

Our study required about an hour of our participants' time, with no expectation of smart home programming experience. Therefore, we cannot make conclusions about these interfaces regarding long-term smart home use. As we mention in Section 2.6, although we emphasized that no experiences with programming nor smart homes was required, many of our participants had programmed before and/or owned IoT devices. Experimenter bias is also possible, but we mitigate this concern by measuring whether participants completed the tasks correctly.

2.6 User Study Results

In this section, we report the results of our user study. Overall, *Rules* worked well for the simple tasks, as did most other interfaces. For the complex tasks, however, *Rules* participants performed poorly while participants using semantic-diff interfaces performed significantly better. The latter participants also found the complex tasks significantly less demanding than participants using *Rules*.

2.6.1 Participants

We received 124 complete responses. We excluded both responses from one participant who did the study twice (and also failed the attention check), seven who gave low-effort or nonsensical answers to the text responses, four who copied and pasted their answers throughout the study, and four who did not complete the tutorial. After applying these criteria, 107 responses remained for analysis.

All participants were between 18 and 74 years old, with 50.5% in the 25–34 range and 28.0% in the 18–24 range. 51.4% of participants identified as women, 46.7% as men, and 1.9% as non-binary. Regarding education, 33.3% held a 4-year college degree, 20.6% had some college background, 15.9% were high school graduates, 11.2% held a 2-year college degree, and 0.9% held a graduate degree. 56.1% of participants had some technical background; they had programmed before, completed a CS-related class, and/or had a CS-related degree or job. 81.3% of participants were not familiar with the IFTTT service [32], but 66.4% of participants owned at least one IoT device.

2.6.2 Correctness

On tasks requiring comparisons of complex variants, we found that participants using the novel semantic-diff interfaces generally outperformed those using *Rules* and *Text-Diff* interfaces (Figure 2.8). Even though participants assigned our novel interfaces completed more tasks correctly than those assigned the control interfaces, the time participants took to complete the tasks did not differ significantly across interfaces. In other words, in the same amount of time, participants were more successful completing the tasks using the semantic-diff interfaces than the control interfaces.

Table 2.4 compares the semantic interfaces and control interfaces by task. In general, *Outcome-Diff: Flowcharts* helped participants identify differences in outcomes when comparing between a few variants and when the variants differed in only a few situations, as in

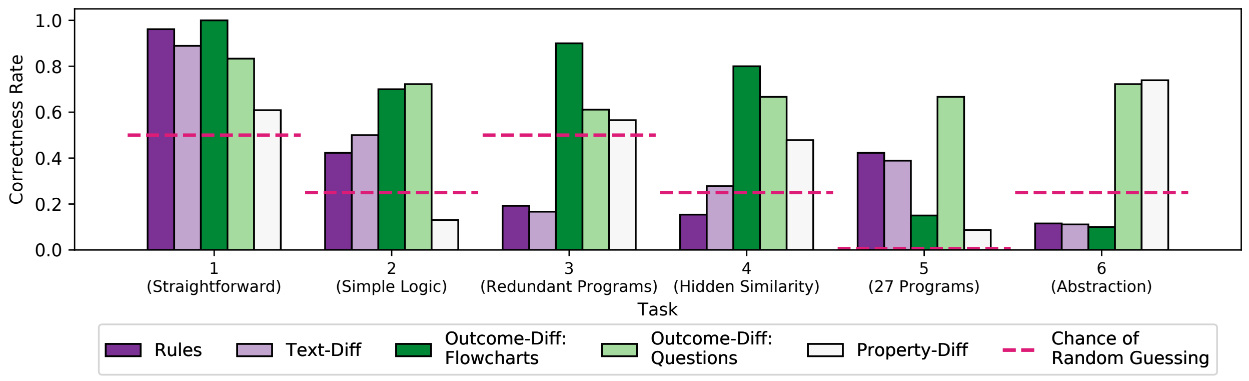


Figure 2.8: Participants' correctness per interface and task. Dashed lines represent chance of random guessing.

Task 3 (Redundant Programs) and Task 4 (Hidden Similarity). When the number of variants or situations became large, as in Task 5 (27 Variants) and Task 6 (Abstraction), *Outcome-Diff: Flowcharts* was less successful. *Outcome-Diff: Questions* was helpful when choosing between few variants, as in Task 3 (Redundant Programs) and Task 4 (Hidden Similarity). It was also helpful even when the variants differed in many situations with many variables, as in Task 6 (Abstraction). For that task, all variables had the same attributes (being on/off) and the goal was abstract enough that it did not matter exactly which variables' state differed. *Property-Diff* was helpful when there were abstract differences across a few variants, as in Task 3 (Redundant Programs) and Task 6 (Abstraction), but not if there were many variants, as in Task 5 (27 Variants), or if the variants had many properties, as in Task 4 (Hidden Similarity).

Compared to the other interfaces, we expected that *Rules* would only work well for Task 1 (Straightforward) and Task 2 (Simple Logic) because they involve short and simple variants. *Rules* participants were significantly more likely to complete the first task correctly than *Property-Diff* participants. As Table 2.4 shows, though, *Rules* did not outperform the semantic-diff interfaces in any other task. *Rules* participants found this first task easy, including P52: "*. . . it made it easy to notice the different modification that was being added to the program.*" A few, however, mentioned that it would be more convenient to see the

Table 2.4: Differences in task correctness between novel interfaces (columns) and controls (rows). The table shows whether a significantly larger (✓) or smaller (×) fraction of participants who used a novel interface answered the task correctly compared to those who used a control. P-values below task names are omnibus tests (Fisher's Exact Test) across all interfaces, while those in table cells are pairwise comparisons. All p-values were corrected for multiple testing using the Holm method.

		Outcome-Diff: Flowcharts	***Outcome-Diff: Questions***	***Property-Diff***
1 - Straightforward	*Rules*			× $p = 0.028$
($p = 0.001$)	*Text-Diff*			
2 - Simple Logic	*Rules*			
($p = 0.001$)	*Text-Diff*			
3 - Redundant Programs	*Rules*	✓ $p < 0.001$	✓ $p = 0.040$	✓ $p = 0.049$
($p < 0.001$)	*Text-Diff*	✓ $p < 0.001$	✓ $p = 0.049$	
4 - Hidden Similarity	*Rules*	✓ $p < 0.001$	✓ $p = 0.009$	
($p < 0.001$)	*Text-Diff*	✓ $p = 0.029$		
5 - 27 Variants	*Rules*			
($p = 0.001$)	*Text-Diff*			
6 - Abstraction	*Rules*		✓ $p < 0.001$	✓ $p < 0.001$
($p < 0.001$)	*Text-Diff*		✓ $p = 0.004$	✓ $p = 0.001$

variants side-by-side. *Property-Diff* participants mostly relied on the "program button" feature (viewing the rules themselves) rather than their assigned interface. This decision is logical because there were no safety properties that could be derived from either variant for the interface to show. While the interface explicitly stated that there were no patterns to show, some participants nonetheless thought the interface was buggy or "*unavailable*" (P61). A few others, such as P43, mistakenly believed that the lack of comparison meant "*the programs were the same.*"

For Task 2 (Simple Logic), we did not find significant differences in correctness between any semantic-diff interface and any control interface. *Rules* participants liked the interface because "*the layout was clear and easy to understand*" (P93). *Text-Diff* participants, like P51, appreciated having the differences highlighted: "*It's great helping me compare programs because it is color coordinated and they are side by side.*" While some *Outcome-Diff: Flowcharts* participants found the interface useful, a few felt ambivalent, like P30: "*The interface allowed me to see the outcomes combined together. Its design did not really help nor hinder my ability.*" Some thought it needed more information, such as for "*separating out what the original program does entirely from what the modified program does*" (P89). On the other hand, *Outcome-Diff:*

Questions participants largely found the interface straightforward and felt it "*helped [them] visualize the scenario*" (P72). *Property-Diff* participants found the interface simple, although a few mistook the properties for the actual rules.

In Task 3 (Redundant Programs), with the exception of *Property-Diff* against *Text-Diff*, participants using the novel interfaces significantly outperformed those using the control interfaces (Table 2.4). Most *Outcome-Diff: Flowcharts* participants found their interface helpful. P26 stated, "*The interface helped, as there were less variables. I could focus on the one situation and identify faults with the second program as there were only two to compare.*" *Outcome-Diff: Questions* participants mostly found their interface straightforward as well, such as P7: "*I think visually showing all the combination was helpful, so I could see easily if the program met them.*" *Property-Diff* participants felt similarly. P48 wrote, "*The interface made it clear that the rules that were desired were only partially met and the interface made this fairly simple to determine.*" A few *Rules* and *Text-Diff* participants found the large number of rules in this task frustrating, but most still found their interface straightforward. P99 found *Rules* "*well structured with a good layout which makes it easy and possible to determine the comparison between both programs.*" For *Text-Diff*, P3 wrote, "*There were a lot of steps to sort through, but the color coding that told me when something was different helped.*"

For Task 4 (Hidden Similarity), *Outcome-Diff: Flowcharts* performed significantly better than the controls, and *Outcome-Diff: Questions* performed significantly better than *Rules*, as shown in Table 2.4. *Outcome-Diff: Flowcharts* participants found the interface mostly straightforward, albeit with a few hurdles. P5 wrote, "*I was able to refer to final configuration of open and closed windows though I had trouble following the "if this situation happens" aspect of it.*" P21 said, "*It was a bit technical but I think I was able to get it.*" *Outcome-Diff: Questions* participants felt confident, such as P6: "*The interface helped significantly, having % match from previous situations let me quickly look to the options and see if any fit the suggested change.*" *Rules* participants found the interface "*a little overloaded*" (P27) or "*cluttered*"

(P31). P24 explained, "*I couldn't see all options at the exact same time to compare them all. I had to memorize and try to figure out the differences.*" *Text-Diff* participants liked the diff highlights, but thought the tasks made the interface confusing. P14 explained, "*It made it very difficult to concentrate and be able to determine the correct answer. Having three windows' statuses being repeated over and over also made it confusing to read.*"

We did not observe significant differences between semantic-diff participants and control participants for Task 5 (27 Variants). However, we found that *Outcome-Diff: Questions* participants outperformed *Outcome-Diff: Flowcharts* participants. Many *Outcome-Diff: Questions* participants, like P7, found the task relatively easy: "*It made it very quick to eliminate options that didn't meet the requirements.*" On the other hand, *Outcome-Diff: Flowcharts* did not reduce the mental load of the task for its participants. P17 stated, "*I had difficulty aligning how the interface views aligned with the requirements. It was not obvious or intuitive to me.*" *Rules* participants, like P27, felt ambivalent about the interface: "*The interface was fine, but it didn't really help to find the result quickly, since I had to click on each program in the dropdown.*" *Text-Diff* participants felt similarly, like P14: "*It was hard to scroll through all 27 programs, and it was a tedious and long process, but simple to understand and visually evaluate.*" The same was true for *Property-Diff* participants, with P96 explaining, "*This one was fairly easy to figure out and the design of the interface did help explain things. You did however have to be careful and watch for what was stated.*"

For Task 6 (Abstraction), *Property-Diff* and *Outcome-Diff: Questions* participants performed significantly better than the control groups (Table 2.4). *Property-Diff* participants mostly thought the interface made the task easy, such as P90: "*It told me exactly yes or no if the program would do what was needed.*" Similar to Task 1 (Straightforward), however, a few participants were frustrated when *Property-Diff* did not show properties for a variant (as it lacked properties by our definition), like P69: "*It's frustrating to see that lack of the always or never rules and makes it feel like there's no program.*" Due to an experimental error,

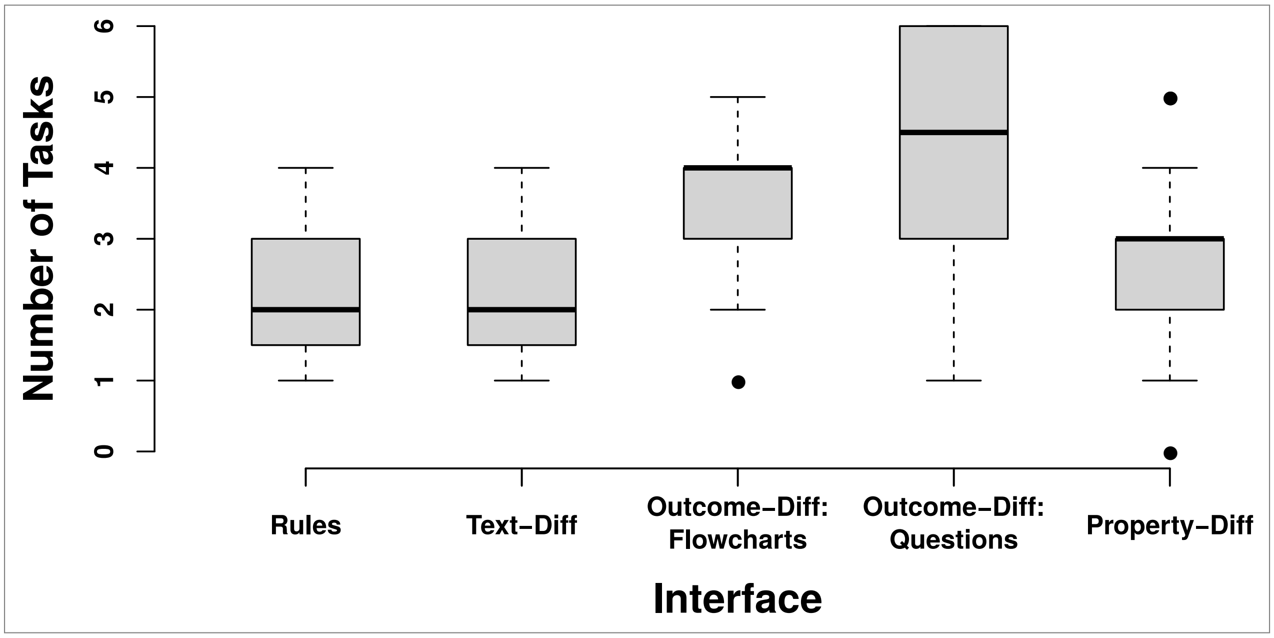


Figure 2.9: The distribution of the number of tasks a given participant answered correctly.

Outcome-Diff: Questions only showed participants 7 out of 21 questions. Nonetheless, the fact that they outperformed the control group raises interesting questions about the design of *Outcome-Diff: Questions*, as we discuss in Section 2.7. Some *Outcome-Diff: Questions* participants, like P7, felt that "*not having to keep track of which options were on and off in each situation made it very easy to quickly determine which combinations met the requirements.*" Some others, however, were overwhelmed: "*There were so many choices that it became cumbersome to figure out*" (P65). Most *Rules* participants found the interface "*okay*" (P76) or "*not very user friendly*" (P35). P24 explained, "*It was hard because of all the information that was presented and the only way to tell the difference between the programs was to either switch back and forth or memorize.*" Many *Text-Diff* participants found the layout to be overwhelming, such as P63: "*When there are too many requirements listed out separately in the "while" section, it can get a little overwhelming to keep it straight.*"

The tasks in this study were not randomly chosen. However, because they cover a variety of different comparison scenarios, we also compared how many tasks each group of partic-

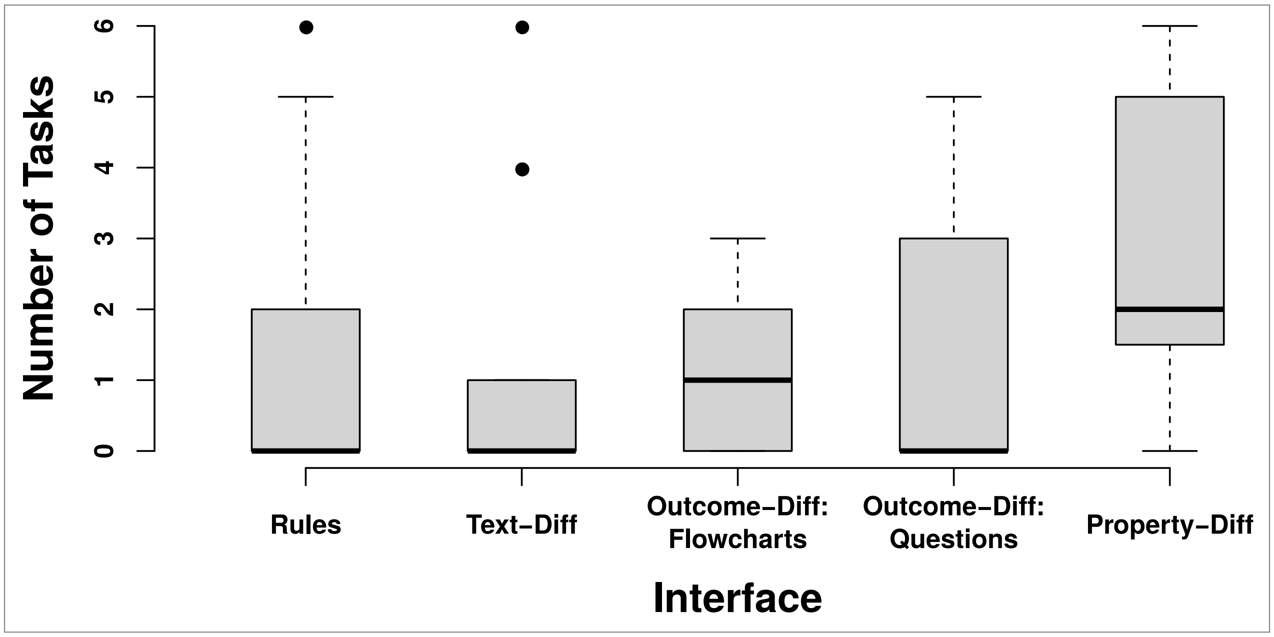


Figure 2.10: The distribution of the number of tasks for which a given participant used the program button.

ipants answered correctly. On average, participants correctly answered 2.9 of the 6 tasks. Overall performance varied significantly between interfaces ($\chi^2(4) = 30.604$, $p < 0.001$), as shown in Figure 2.9. Participants using the two interfaces with low-level information, *Outcome-Diff: Flowcharts* and *Outcome-Diff: Questions*, completed significantly more tasks correctly than those with other interfaces (vs. *Rules*: $p < 0.001$ and $p = 0.010$, respectively; vs. *Text-Diff*: $p = 0.006$ for both; vs. *Property-Diff*: $p = 0.014$ and $p = 0.011$, respectively).

2.6.3 Reliance on Seeing Programs

On average, participants used the "program button" feature to see the rules themselves for 1.6 of the 6 tasks. The number of tasks for which participants chose to look at the programs differed significantly between interfaces ($\chi^2(4) = 12.024$, $p = 0.017$), but we did not find any significant pairwise differences (Figure 2.10). In our Poisson regression, we observed that *Property-Diff* participants were more likely to consult this feature than our baseline *Rules* participants ($p = 0.004$), while women were more likely to do so than men ($p = 0.007$).

Furthermore, participants without any college education were more likely to consult this feature than participants with a graduate degree ($p = 0.049$).

Half of the *Rules* participants, who already saw the rules themselves in their interface, did not rely on this feature. Unexpectedly, though, some relied on both this feature and the interface to compare variants more easily or to show long programs on the same screen. A few relied only on this feature; one found its text easier to read than the three-column layout. Many *Text-Diff* participants relied on the interface alone because it was easier to use, but some found both the interface and the program button helpful.

Almost all *Outcome-Diff: Flowcharts* participants relied on the interface for the most part because they preferred seeing the outcome differences. P30 stated, "*It's easier to just focus on the outcomes that way and see whether it meets the conditions or not. Also, the interface is a visual compared to the 'Programs' which is just text.*" Three participants used both to give a more complete picture of the task, while one participant relied mostly on the programs because it gave more information. Most *Outcome-Diff: Questions* participants stated that they generally did not rely on seeing programs because the interface was easier to use. A handful relied on both the programs and the interface in order to get a more complete picture, while two relied mostly on the programs.

Most *Property-Diff* participants relied on the interface alone or with the programs. They found the interface easier to grasp while the programs had unnecessary information. Some mentioned that this preference varied between tasks.

2.6.4 Perception of Interface Usability

SUS scores did not differ significantly across interfaces. The mean score per interface ranged from 53.8 to 69.9, while the median ranged from 54.2 to 73.3. An SUS score of 68 is often considered average. Some interfaces' mean and median scores were at or below 68, though the difficulty of our tasks could have biased participants' perceptions negatively. We designed

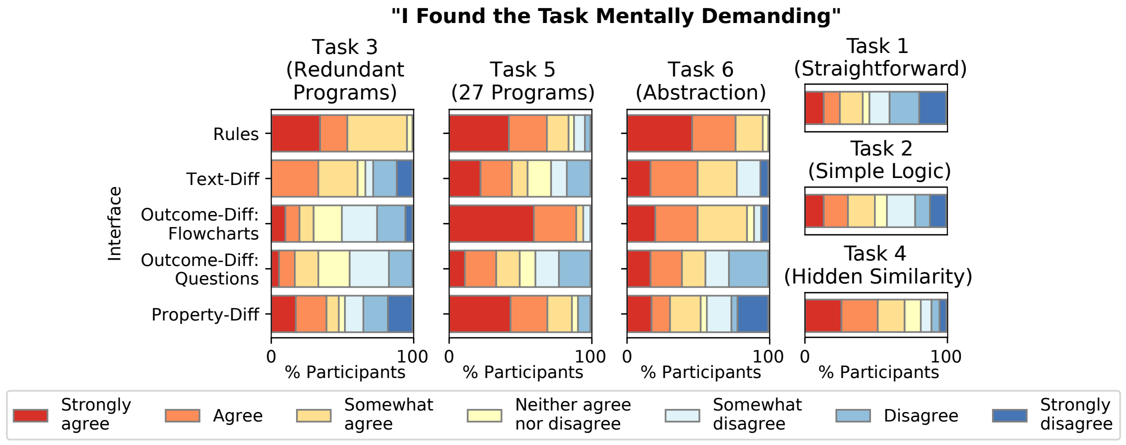


Figure 2.11: The distribution of participants ratings, by interface, for how demanding they considered each task. For Tasks 1, 2, and 4, ratings did not vary significantly by interface, so we present only the aggregate distribution for all participants.

the tasks to highlight both benefits and drawbacks of each interface, so all participants had to complete tasks for which their interface was unhelpful. *Outcome-Diff: Flowcharts* participants were also significantly more likely to complete some of the complicated tasks correctly than control participants (Section 2.6.2), despite the low SUS rating of this interface.

2.6.5 Study Experiences

In total, participant perception of how demanding the tasks were differed significantly ($\chi^2(4) = 12.319$, $p = 0.015$), with *Outcome-Diff: Questions* participants finding tasks significantly less demanding compared to *Rules* participants ($p = 0.014$). This perception differed significantly between interfaces for Task 3 (Redundant Programs) ($\chi^2(4) = 18.987$, $p = 0.005$), where *Text-Diff* ($p = 0.033$), *Outcome-Diff: Flowcharts* ($p = 0.001$), and *Outcome-Diff: Questions* ($p = 0.001$) participants all found the tasks less demanding than *Rules*. For Task 5 (27 Variants) ($\chi^2(4) = 18.375$, $p = 0.005$), *Outcome-Diff: Questions* participants found the task less demanding than *Outcome-Diff: Flowcharts* participants ($p = 0.002$). For Task 6 (Abstraction) ($\chi^2(4) = 18.834$, $p = 0.005$), *Outcome-Diff: Questions* ($p = 0.015$) and *Property-Diff* ($p = 0.003$) participants found the task less demanding than *Rules*. Figure 2.11 shows participant ratings of how demanding tasks were.

Although participants were more likely to answer tasks correctly using the novel interfaces than the control interfaces, participants' confidence in their answers did not vary across tasks, either for any individual task or in aggregate across tasks. While they were not significantly more likely to answer tasks correctly, male participants were significantly more confident in their answers than female participants ($p = 0.044$) in our regression model.

Similarly, user perception of interface helpfulness did not differ significantly overall or for any task. Emphasizing the potential benefit of user interfaces that support TAP, participants without a technical background found the interfaces more helpful than those with such a background ($p = 0.017$) in our regression model. Furthermore, participants who did not own IoT devices found the interfaces more helpful than those who did ($p < 0.001$).

2.7 Discussion and Conclusion

To help users reason about differences in TAP programs, we designed a set of complementary interfaces that contrast TAP variants at different levels of semantic abstraction. We conducted an online user study to compare semantic-diff interfaces (*Outcome-Diff: Flowcharts*, *Outcome-Diff: Questions*, and *Property-Diff*) to two traditional interfaces (*Rules* and *Text-Diff*) in helping users complete TAP tasks with different characteristics. In our online user study, we found that participants could correctly reason about differences between variants of short, simple programs by examining only the rules themselves. However, when comparing variants of long, complex programs, participants using semantic-diff interfaces significantly outperformed those using *Rules* or *Text-Diff*. *Outcome-Diff: Flowcharts* participants performed better when the task required identifying a manageable number of situation-specific differences in complex programs, while *Property-Diff* participants performed better when reasoning about more abstract differences. Participants using the low-level interfaces, *Outcome-Diff: Flowcharts* and *Outcome-Diff: Questions*, were able to identify program differences correctly across a wider variety of tasks than other users. *Outcome-Diff: Questions* participants often found tasks less demanding than others and significantly outperformed *Outcome-Diff: Flowcharts* participants when comparing many variants.

Our work advances intuitive methods of surfacing smart-system behaviors to users. We show that TAP interfaces should visualize information at multiple levels of granularity. By having automated reasoning using formal methods underpin user interfaces for end-user programming, the community can help TAP users better match a system's behaviors to their intent. **To facilitate future work, our open-source code for the interfaces, survey instrument, and regression tables are available online [96]**.

2.7.1 Deployment Recommendations

Our interfaces have complementary strengths and weaknesses based on the characteristics of the TAP variants being compared. We believe that real-world deployments should take these trade-offs into account, showing the user a diff interface appropriate for particular situation and set of variants they are comparing. Fully understanding how to approximate user intent and identify the relevant characteristics of the TAP variants in order to automatically select a contextually appropriate diff interface requires future research, as does better understanding the potential usability confounds of showing a single user different interfaces based on the situation. Nonetheless, our results provide initial suggestions for contextually appropriate interfaces. When a user is comparing variants of short programs (each with a few rules and each rule with a few conditions), they should simply view the rules themselves. As programs becomes longer and more complex during the modification process, the system should present semantic-diff interfaces to help users understand the effects of their modifications. Because *Outcome-Diff: Questions* helped participants in our study reason about a wider variety of tasks and made the tasks less demanding, it could perhaps be presented by default. If the user is choosing from many variants, an interactive form-based interface like *Outcome-Diff:*

Questions will help them eliminate undesirable choices faster.

Rather than trying to automate the selection of an appropriate interface, the system could instead ask the user whether they want to see differences in situation outcomes or high-level trends (properties). To account for *Property-Diff* participants' reliance on viewing the rules via the program button in our study, *Property-Diff* interfaces should also provide prominent access to the rules themselves, perhaps even displaying them by default. More broadly, we found that participants sometimes struggled to understand how the properties and situations the semantic interfaces displayed related to the trigger-action rules. We recommend that real-world deployments further clarify this connection, perhaps in an expanded tutorial introduction before walking the user through concrete interfaces.

Our interfaces could also perhaps be useful during program creation. To minimize repeated computation and facilitate just-in-time diffs after each change, future work should incrementally build transition systems rather than producing a new one from scratch each time. Future work can also consider applying these interfaces outside of trigger-action programming, such as to outcome-based program synthesis or constraint-based programming.

CHAPTER 3
SUPPORTING END USERS IN DEFINING REINFORCEMENT-LEARNING PROBLEMS FOR HUMAN-ROBOT INTERACTIONS

3.1 Introduction

Reinforcement learning (RL) is applicable to a variety of domains [83, 7, 36], but end users without sufficient technical expertise may have trouble applying it to their own needs. They may face challenges such as determining how to train the RL agent efficiently and how to define and administer rewards. Existing work has attempted to support end users by incorporating user feedback through policy shaping, guided exploration, and augmented value functions [4].

While most work supports end users in training the RL agent on a defined task, little work has attempted to address end-user challenges in *defining* that task. It may appear straightforward to define an RL task as a Markov decision process (MDP) with a set of states, a set of actions, and a set of rewards. However, users might find it difficult to anticipate the appropriate state space and action space. Specifying too many states or actions can cause the agent to learn over an excessively long period of time. Specifying too few, on the other hand, may cause the agent to learn a suboptimal policy.

We propose a tool for assisting end users in defining and refining RL problems. Our tool consists of a graphical interface that guides the user in defining RL problems both before and after training an agent. Using built-in primitives, users will first specify the states, actions, and rewards of the MDP. Our tool walks the user through this process in an accessible manner. The RL agent will then train on the resulting MDP. (For our tool, we assume a Q-learning agent trained on episodic tasks.) When training is complete, our tool shows the user a simulation of the agent interacting with its environment based on the learned policy. The user can then evaluate whether their MDP appropriately captures the intended task. They can accept the final policy, revise it, or revise the MDP definition and try again.

Our work focuses on RL applications in Human-Robot Interaction (HRI), which concernssubjects social interactions between people and robots. In HRI tasks, robots must behave in a socially appropriate manner by adapting to personal preferences, cultural norms, and other context-specific factors [2, 91, 37, 39]. One example in HRI is the issue of robot abuse. Multiple public incidents have occurred in which people purposefully blocked the robots' path or sensors, hurled obscenities against them, or resorted to violent behaviors like kicking and hitting [64, 11, 76]. Robots designed to accomplish tasks unsupervised must somehow mitigate these incidents.

How can a user help the robot navigate complex social interactions, such as avoiding or pacifying incidents of robot abuse? Programming languages like Python for the Pepper robot[1] require substantial technical knowledge. End-user programming paradigms and systems, such as Choreographe [72], demands less technical knowledge but still requires the user to anticipate the ideal robot behavior under a specific circumstance for all possible circumstances. A similar problem arises for Learning from Demonstration (LfD).

In contrast, an RL approach can help the robot adapt to the many scenarios it will encounter without requiring the user to detail each possibility, making it especially well suited to those with minimal programming and domain expertise. Unfortunately, to leverage RL for this problem, the user still needs advanced knowledge of the environment. In our example, users may neglect to realize that states related to bystanders may affect the likelihood of robots being bullied, choosing instead to only consider states that relate to the bullies and the robot. Our tool seeks to address this gap, with a focus on HRI tasks but may also be extended to other applications.

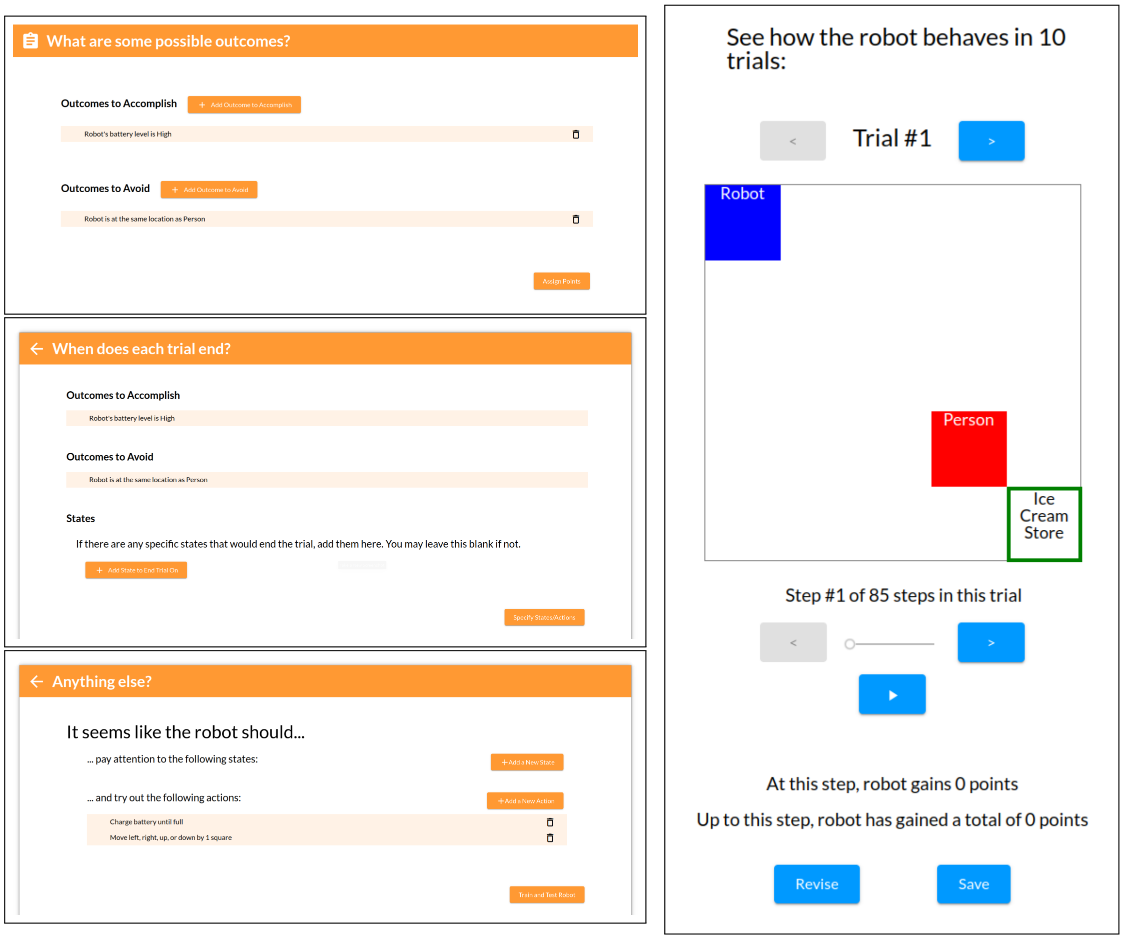
1. `https://www.softbankrobotics.com/emea/en/pepper`

3.2 Vision for the Interaction

We envision a multi-stage interaction between the user and the RL system. We present our design here, with some parts implemented (Figure 3.1). Via a graphical interface, the user will first specify the states, actions, and rewards of the MDP (Figure 3.1 left) by selecting from a set of system-defined choices (Figure 3.2). It may be unclear to the user how these components of the MDP relates to the learning process, e.g. what aspects of the environment should be specified as part of the state space. Our interface clarifies this by first asking users to specify outcomes the agent should try to accomplish or avoid. These will be considered state features that, when achieved, will positively or negatively reward the agent. The interface will also assume these outcomes to be terminating conditions for an episode, and ask users to modify them or specify additional terminating conditions as needed. Lastly, the user is given the opportunity to specify any additional states or actions of the MDP.

In addition, some users may find it difficult to think about RL tasks in the terms of MDP. Our design offers these users the option to, instead, define the task by predicting parts of the optimal policy. They will describe these predictions in the form of IF-THEN (trigger-action) statements, and the system will extract a potential MDP based on these statements. For example, if the user writes "IF a person starts hitting the robot THEN the robot should run away," the system will infer that "whether there is a person hitting the robot" is a relevant state and "robot running away from people" is a relevant action for the MDP overall.

Training the robot will produce information for debugging the MDP. During training, our interface design will show the policy of the robot at each timestep as well as information such as the reward accumulated for each episode. The user can monitor this to understand the trends in the robot's learning, such as whether the robot is maximizing its rewards in the first place. After training, the system will simulate executing the robot based on the learned policy for a set of episodes, and visualizes this to the user along with statistics such as the rewards gained for an episode.


What are some possible outcomes?

Outcomes to Accomplish

Robot's battery level is High

Outcomes to Avoid

Robot is at the same location as Person

When does each trial end?

Outcomes to Accomplish

Robot's battery level is High

Outcomes to Avoid

Robot is at the same location as Person

States

If there are any specific states that would end the trial, add them here. You may leave this blank if not.

Anything else?

It seems like the robot should...

... pay attention to the following states:

... and try out the following actions:

Charge battery until full

Move left, right, up, or down by 1 square

See how the robot behaves in 10 trials:

Step #1 of 85 steps in this trial

At this step, robot gains 0 points

Up to this step, robot has gained a total of 0 points

Figure 3.1: Screenshots showing selected stages of the user interaction with our graphical interface. Top left: The user can first specify outcomes to achieve (positively reward) or avoid (negatively reward) as part of the MDP. Center left: The user can then specify terminating conditions for an episode. Bottom left: The user can specify any additional states or actions that were not covered in the previous screens. Right: After the system has trained and tested the agent with the user-defined MDP, it will show the user simulations of the agent's behavior, along with information such as accumulated rewards to help the user debug.

State Features

- **Person's Location**: position and distance relative to the robot;
- **Person's Mood**: happy, engaged, or frustrated;
- **Person's Actions**: whether they are looking at the robot or speaking, what they are saying, or other user-specified actions;
- **Miscellaneous**: what the robot is doing, is the environment noisy, is there trash around the robot.

(Robot's) Actions

- **Gaze**: look at a specific person, object, or somewhere else;
- **Speech**: say a specified phrase, stop speaking;
- **Indicate**: motion to people or objects;
- **Navigate**: start/stop traveling a specified distance toward a specified direction at a specified speed;
- **Miscellaneous**: rest, shut down.

Outcomes to Reward

- **Person's Mood**: happy, engaged, or frustrated;
- **Person's Speech**: whether they refer to the robot positively, whether they say certain phrases;
- **Person's Actions**: whether they acknowledge or ignore the robot; whether they respond as expected (e.g. whether they are following the robot's directions).

Figure 3.2: A work-in-progress list of interface primitives we plan to support, deduced from reviewing existing tasks studied in HRI literature (e.g. [77]). "Person" refers to a single individual interacting with the robot, such as a conversation partner.

To illustrate a use case of our proposed tool, we walk through the following example. Suppose a robot is deployed at a store and tasked with maximizing average customer satisfaction, as measured by an in-store exit survey. A user who is comfortable with the ideas of MDP might use our interface to specify state features related to the mood of the customers. They might specify actions such as greeting customers and approaching customers to offer assistance. They might consider each customer to be an episode, and the positive or negative survey rating of the customer to be the reward for that episode. A different user might choose to to define the tasks by writing IF-THEN statements such as "IF a customer begins to get frustrated THEN approach the customer to offer assistance," allowing the system to extract a similar MDP as above.

Based on the MDP, the tool might recommend additions to the states, actions, and rewards to account for potential robot abuse. For example, it might recommend the age of the customer as a state feature since children might be more unruly and more likely to hinder the robot, and the ability to run away as a possible action. The user decides to accept these changes, and then train the robot. When training has concluded, suppose the user sees that the final policy has the robot walk away from people intentionally blocking its path. The user wants to know if it will be more effective in raising customer moods and reducing antagonistic encounters if the robot instead asks the customer to politely move, which was not an action included in the original MDP. They can, therefore, return to the starting MDP, modify it to add this new action, and re-train the robot.

3.3 Conclusion

A large body of work has examined supporting end users and experts in improving their RL system after it is deployed, yet little has attempted to help at the very beginning of this lifecycle. Our work proposes supporting end users in defining RL problems through structured templates and iterative debugging.

CHAPTER 4
ROBOT MEDIATION OF PERFORMER-AUDIENCE DYNAMICS IN LIVE-STREAMED PERFORMANCES

4.1 Introduction and Background

In an in-person performance, performers receive audience input such as applause, cheering, booing, or silence. By acknowledging and responding to this input, the performers allow the audience to influence the performance. In a live-streamed performance, where the audience is watching from afar in real time, such interactions are severely muted. For example, on Twitch, viewers (audience) can type in the chat and tip money to the streamers (performers). Yet while the streamers can acknowledge these feedback, they cannot see the viewers individually. Consequently, performers cannot receive as much feedback nor as quickly as in an in-person performance, reducing engagement and impact from the audience.

Robots might strengthen the interactions between performers and audience members in virtual performances by supporting accurate and expressive communication between them. As a mediator, the robot can leverage its physical and social features to actively prompt for audience participation, respond to them, and express their sentiments to the performers. Prior work has explored the potential for leveraging robots as theatre actors [25, 33], comedians [90, 57], musicians [10, 28, 29], and audience companions [26, 27]. It has not yet examined how robots can serve as performance mediators. Beyond HRI, prior work has explored the design implications for live-streamed activities [48, 49, 79, 78], but has not yet explored the effects of a physical robot in this setting.

In this work, we conducted a between-subject study to investigate robot mediation in live-streamed performances. Since performers will likely choose chatbots for online technology mediation, our research question was: **Compared to a chatbot mediator, can a robot mediator enhance the performer-audience connection in a live-streamed,**

remote performance? With a robot mediator, we hypothesized participants would feel more included, perceive the performers to be more responsive, and enjoy the experience more.

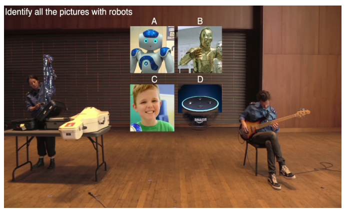
We streamed three interactive musical performances on Twitch and recruited participants to join one of them. During each performance, participants answered prompts in the chat. Their responses were randomly chosen to inform the rest of the performance. In one performance, a chatbot employed text-to-speech to react to the audience and to communicate the selected responses to the performers. In the other two performances, a robot in the performance space replaced the chatbot to speak while gesturing. We surveyed the participants between breaks and after the performance about their experience. We also recorded the chat history and interviewed the musicians.

We did not find significant differences in audience experience between the chatbot performance and the robot performances. However, based on survey responses and chat messages, some participants found the robot helpful in fostering engagement. We reflect on the experience of having a performance robot mediator and highlight design considerations.

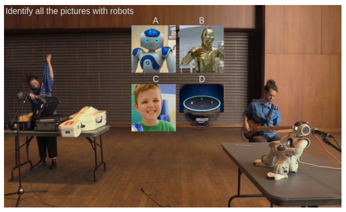
4.2 Methods

To distinguish the robot mediator's influence on audience engagement, we sought to maximize the audience's unfamiliarity with the experience. Thus, for the performance, we chose experimental music over familiar art forms like pop music or improvisational comedy. We chose an interactive experimental piece [23] (detailed in Section 4.2.2) that involved technology mediation by prompting audience for suggestions, based on which musicians must improvise to musically respond.

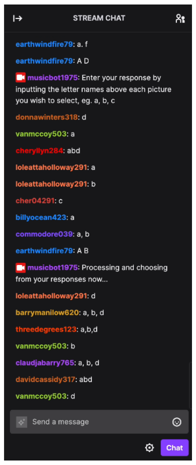
We streamed three live performances of this composition. In all three, a mediator named "Musicbot1975" prompted for and reacted to audience responses. It also informed performers of the responses. In the baseline condition/performance, Musicbot1975 was a text-to-speech



(a) Screenshot of the *Chatbot* performance.



(b) Screenshot of a *Robot* performance.



earthwindfire79: a. f

earthwindfire79: A D

musicbot1975: Enter your response by inputting the letter names above each picture you wish to select, eg. a, b, c

donnawinters318: d

vanmccoy503: a

cheryllyn284: abd

loleattaholloway291: a

loleattaholloway291: b

cher04291: c

billyocean423: a

commodore039: a, b

earthwindfire79: A B

musicbot1975: Processing and choosing from your responses now...

loleattaholloway291: d

barrymanilow620: a, b, d

threedegrees123: a,b,d

vanmccoy503: b

claudjabarry765: a, b, d

davidcassidy317: abd

vanmccoy503: d

(c) Example snippet of the Twitch chat.

Figure 4.1: Audience view of the performances on Twitch.

chatbot. In the other two performances, Musicbot1975 was a NAO robot in the performance space. The robot spoke the same text as the chatbot, but also faced the party they were speaking to (camera or the musicians) and gestured. It moved autonomously when at rest. In both cases, Musicbot1975 presented participation prompts in chat rather than aloud to avoid excessive audio interruptions.

We placed the video camera in front of the performers, and captured with microphones the audio of Musicbot1975 and the musicians. Figure 4.1 shows what the audience saw on Twitch. For the robot performances, the robot sat on a table to the front left of the musicians. This layout allowed the camera to capture both the musicians and the robot, and the musicians to see the robot while performing to the camera.

4.2.1 Procedure

Participants signed up for one performance and, at the assigned time, logged onto Twitch with an account created by us. Each performance contained three movements, each of which lasted 10-15 minutes with a unique theme. Participants completed a short survey after

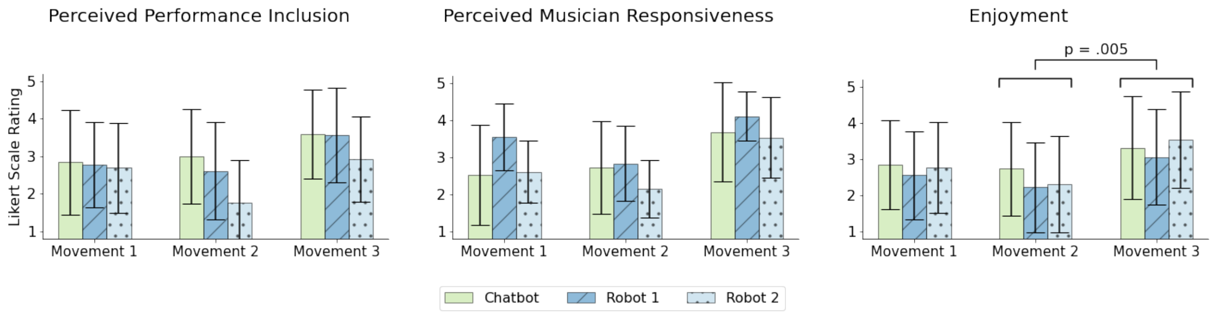


Figure 4.2: The recruited audiences enjoyed Movement 3 more than Movement 2. Beyond this, their experiences did not differ significantly between conditions or between most movements.

each movement. The survey asked for their perceived performance inclusion (the extent to which they felt included in the performance), perceived musician responsiveness (the extent to which they felt the musicians were responding to their input), enjoyment level, and perception of Musicbot1975. We recorded all chat messages.

At the end, we gave participants a similar survey as before about the overall performance. We administered the Robotic Social Attributes Scale (RoSAS) [12] regarding Musicbot1975. We measured extroversion with a 5-point variant of the Extroversion scales in the Revised Eysenck Personality Questionnaire (EPQ R-A) [21], with 5 indicating strong agreement with extrovert statements or strong disagreement with introvert statements. We asked for demographics and their prior experience with Twitch. We also interviewed each performer about their experience. The UChicago IRB approved this study.

4.2.2 Performance and Robot

The robot (or chatbot) verbally introduced each movement while looking and gesturing toward the camera. To react to the audience, we programmed robot gestures to mimic happiness (waving hands above its head), sadness (twisting its wrists in front of its eyes as if crying), confusion (a hand scratching the side of its head), and anger (shaking its fists up and down).

In Movement 1, Musicbot1975 prompted in the chat for audience suggestions, e.g. “Name a composer” or “Name your favorite food.” Prompts occurred every 30-90 seconds in context of the performance narrative. Suggestions were randomly selected to fill in template sentences that Musicbot1975 then read aloud, such as “Covering <Mozart> with <sushi>.” We used an existing emotion classifier [19] to output whether the selected response conveyed happiness, sadness, confusion, or anger. The robot reacted accordingly and instructed the musician (an oboist) to express this emotion in their improvisation. Movement 1 concluded with a notification that hackers had supposedly invaded the performance stream.

In Movement 2, multiple-choice questions were superimposed on top of the stream about every minute (Figure 4.1). The questions asked participants to select from a group of pictures all those that satisfied a particular criteria, e.g. “Identify all pictures with robots.” Musicbot1975 described the questions as CAPTCHA security checks to help identify the hackers. A random audience response was chosen for each question. If the response was correct, the robot would react with the “happy” gesture, and the electronic accompaniment provided to the musicians would sound calmer and more pleasant. If it was incorrect, the robot would react with the “sad” gesture, and the electronic music would sound more agitated and dissonant. Two musicians (cellist and electric bassist) performed, with the former at times playing a cello, dancing, and pretending to inspect a suitcase while the latter played the bass. They changed their improvisations in response to changes in the electronic accompaniment. Movement 2 ended with the robot declaring a successful neutralization of the hacking attempt.

In Movement 3, Musicbot1975 hosted a lip sync contest for celebration. All three musicians from before competed for audience votes. Audience members were prompted to give pre-scripted commands to the performers, e.g. “Play faster” or “Play higher.” Responses were tallied for each command. Once the tally passed a certain threshold, the robot read the command aloud to the performer. For example, after five participants selected “Play faster,” the robot read aloud the command, the electronic accompaniment played at a faster tempo, and the musicians played noticeably faster in response. The robot coupled verbal instructions with gestures, such as raising its hands for "Play higher." The audience voted for their favorite performer after each round of the contest, and the robot congratulated or dismissed a contestant accompanied by a "happy" or "sad" gesture. The performance concluded with the ultimate winner symbolically becoming a robot.

4.2.3 Participants

We recruited participants online through the Center for Decision Research Virtual Lab at University of Chicago Booth School of Business. Eligible participants were above the age of 18, lived in the United States, and had access to a computer/laptop with reliable internet connection. We did not filter for Twitch users nor music experience in order to elicit insights generalizable to most users. Participants were paid $12 with additional amount prorated by the time they spent on the study, which averaged to be about 90 minutes.

Our study had 19 participants in the *Chatbot* audience, 18 in the first robot audience (*Robot1*), and 13 in the second (*Robot2*). Everyone was between 18-74 years old, with 46% of the participants between 18-24 years old and 24% between 25-34 years old. Most (82%) identified as women while the other 18% identified as men. For their highest education completed, 36% of the participants held a 4-year college degree, 32% underwent some college, and 30% received an advanced degree. Most (78%) did not hold a job, earn a degree, or major in technology-related fields. Most were not avid Twitch users: in the six months prior, 86% had never streamed on Twitch and 50% had never viewed Twitch streams.

4.3 Results

We analyze each robot performance as its own separate condition, resulting in three conditions total. This is due to differences between *Robot1* and *Robot2*. *Robot1* occurred in the morning, while *Chatbot* and *Robot2* occurred in the afternoon. Furthermore, in *Robot1*, there was a longer delay (10 extra minutes) before Movement 2 than expected due to a technical difficulty, which was resolved before proceeding. Although this delay likely did not markedly impact audience experience, we decided to account for it as a potential confound regardless.

For quantitative data analysis, we performed a 3 (audience) x 3 (movement number) mixed analysis of covariance (ANCOVA) test where the audience type was a between subjects factor (*Chatbot*, *Robot1*, or *Robot2*) and the movement was a within subjects factor. We performed this test with age, gender, education level, tech experience, experience in viewing or streaming Twitch in the last six months, and extroversion as covariates. For chat message volume, we could not match audience usernames with their survey responses, and therefore conducted a mixed analysis of variance (ANOVA) without covariates. If the test determined a groupwise difference, we followed up with Tukey's HSD test to determine significant pairwise differences. For Likert-scale questions regarding the overall performance, we compared between groups using the ANCOVA with the audience type as a fixed factor and with the same covariates as before. Effect sizes for the ANOVA tests are reported as partial eta squared (η^2).

4.3.1 Audience Experience

For each of our movement-specific metrics (perceived performance inclusion, musician responsiveness, and enjoyment), a two-way mixed ANOVA revealed that the main effect of the condition was not significant, and that the interaction effect between the conditions and the movements was not significant. There was a significant main effect of the movements for enjoyment ($F(2, 76) = 3.67, \eta^2 = .04, p = .030$), and a Tukey's HSD test revealed that participants enjoyed Movement 3 more than Movement 2 ($p = .005$) (Figure 4.2). We did not find any significant pairwise differences between metrics on overall performance perception.

Most participants, regardless of condition, noted in the survey that they found Movement 3 to be the most engaging. Compared to previous movements, they felt more included and found the musicians more responsive. Some explicitly pointed out that the latter positively influenced the former. A *Chatbot* participant noted: “This section was a lot better as the music changed based on the audience. I was much more engaged.” Most participants recognized how music changed based on audience responses, sometimes citing the pre-scripted commands. Multiple participants found this movement more intuitive than Movement 2, in which the correct CAPTCHA answers were not always obvious. A *Chatbot* participant found the voting system “quite straight forward... It didn’t require too much thinking like the captcha round nor did it feel super random like the first round.”

Chat activity suggested *Robot2* participants were more engaged with the performance and with each other than the other audiences. With average number of messages sent per participant as the dependent variable, a 3x3 mixed ANOVA revealed significant interaction effects between the movements and the conditions ($F(4, 98) = 5.98$, $\eta^2 = .20$, $p < .001$), but no significant main effects of either factor, nor significant pairwise differences between levels of each factor with Tukey’s HSD tests. As the performance went on, *Robot1* participants sent less messages on average while *Robot2* participants sent more messages on average (Figure 4.3). Both *Chatbot* and *Robot1* participants kept to themselves, responding only to the prompts and rarely commenting otherwise. However, multiple *Robot2* participants commented and joked with each other, starting with two in Movement 1. As the performance went on, more participants joined in and the comments became more frequent. As one participant noted: “People opened up a little more and were typing more. I noticed the same few people typed a lot more than others.” Chat messages directly referenced Musicbot1975 (“[P1]: youre a funny little robot”, “[P6]: the robot is stressed”), the performance props (“[P2]: why a silver boot / [P3]: theseoutfitsaresnazzy”), the performers (“[P2]: how are they not laughing”), the performance theme (when a musician was “converted” into a robot in

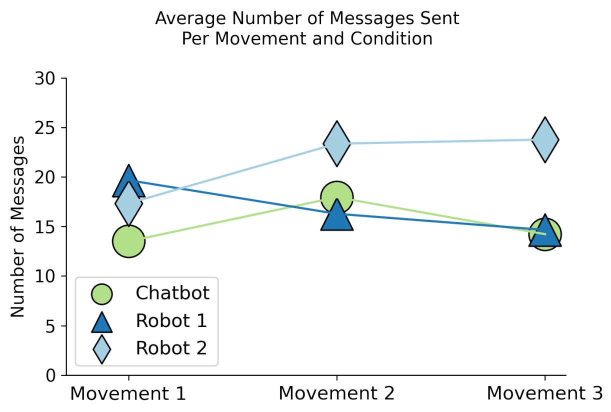


Figure 4.3: *Robot2* participants on average sent more messages as the performance went on, unlike the other two audiences.

Movement 3: "[P4]: black mirror is taking notes / [P1]: This is a religion / [P5]: a cult"), and each other ("[P1]: will you marry me musicbot / [P6]: i vote [P1] off the island").

As noted earlier, we could not account for covariates for chat activity because we could not match the chat usernames with the survey responses. For context, *Robot2* participants self-reported to be the least extroverted (M = 2.45, SD = 1.14), followed by *Robot1* participants (M = 2.90, SD = 1.28) and lastly *Chatbot* participants (M = 3.42, SD = 1.24). *Robot2* participants also skewed younger than the other two audiences, with 62% between 18-24 years old and completing at most some college at the time of the study. They may have been more tech-savvy and receptive, and thus more active in chat.

4.3.2 Audience Perception of the Performance Mediator

Regarding Musicbot1975, we did not identify significant differences between the three groups of participants on the RoSAS scale [12]. On the 9-point scale (with 9 corresponding to strongly agree to items indicating competence, warmth, or discomfort): Participants found Musicbot1975 to be moderately competent, with the *Chatbot*, *Robot1*, and *Robot2* participants rating competence items 5.11 (SD = 2.15), 5.37 (SD = 1.66), and 4.29 (SD = 2.20) on average, respectively. They did not find it very personable, rating warmth items 3.41 (SD =

2.25), 3.90 (SD = 2.22), and 3.03 (SD = 1.89) on average, respectively. They did not find it discomforting, rating discomfort items 4.35 (SD = 2.82), 4.45 (SD = 2.57), and 4.58 (SD = 2.76) on average, respectively.

In Movement 1, two *Robot1* participants thought the music changed based on audience response precisely because of the robot present, with one stating: "In the beginning the music bot asked us what genre of music and then a robot said it out loud to the player. I would like to think the player tried to play something based on their interpretation of our answers." A *Robot2* participant noted: "The musicbot did read out some chat messages out loud which was a nice touch, it made the performance feel more interactive." Some participants found the robot's presence positive while some did not comment or did not find it interesting. In Movement 3, *Robot1* participants described the robot as "calm," "pleasantly sitting on the table," or "more polite and objective." One *Robot2* participant thought "The bot talked and moved in rhythm with music," even though we did not program the robot to move in rhythm, but rather it moved autonomously when resting.

4.4 Conclusion and Discussion

In this work, we held three performances live-streamed on Twitch with a chatbot or a robot mediating performer-audience interactions. Although we did not find significant differences in audience experiences between mediators, some participants pointed to the robot as a reason for higher engagement. We review some of the lessons we learned from designing the robot mediator and from interviewing the musicians.

How much of the audience experience should be revealed to the performers? One musician wanted to see the chat projected in front of the room. Another wondered what the audience was seeing, as some of the performative elements (e.g. CAPTCHAs) were superimposed onto the stream but not shown to the performers. The ideal mediation system would need more than just a robot to share audience experience.

How should the performers depend on the robot? To support the musicians, we projected in front of them the sheet music they consult for improvisations and the captions for the chatbot/robot. This made the performance more comfortable for them, but also weakened their reliance on the robot. The musicians also thought that the robot did not necessarily help their improvisations, but rather gave instructions that they needed to follow. It may have even hindered their improvisations slightly by distracting them with movements.

What utility does a mediator, especially a robot one, bring to the performance? Experimental musicians may find robot mediation rewarding, but the experimental nature of this genre can confuse the audience. Future work should study how robots can enhance this distinctive setting. Mediators may also benefit other, participatory genres of music, such as cabarets in which entertainers perform to audiences sitting at tables, and gospel music in which a cantor leads the audience to sing in a call-and-response manner. It is unclear whether a robot mediator can leverage its robot form to uniquely benefit these performances. Lastly, our work may help Twitch streamers discern how to integrate a robot into their live performances.

CHAPTER 5
EFFECT OF VOICE EXPLANATIONS AND ANTHROPOMORPHISM ON AGENCY ATTRIBUTION AND TRUST IN BODY-ACTUATING DEVICES

Body-actuating devices, when attached onto a user, apply force, motion, and vibrations to affect the user's sense of touch and movement. Examples include robotic exoskeletons [24, 97] and electrical muscle stimulation (EMS) [85, 43]. They can enhance users' physical capabilities [62]. They can also convey information to the user, such as by enabling haptic feedback in virtual reality [46] and mixed reality [47], training motor skills [85, 63], and communicating dynamic affordances of objects [45]. In particular, an advantage of body-actuating devices is that they can protect users from experiencing harm before the user themselves recognize it and react. For example, it can prevent the user from unintentionally burning their hand on a hot object [45].

Users may hesitate to adopt such systems, as they reduce user agency or sense of control [42, 59]. One of Shneiderman's eight golden rules of user interfaces states that they should "support internal locus of control" because users "strongly desire the sense that they are in charge of the system and that the system responds to their actions" [80]. Automation systems such as body-actuating systems, however, work against this [65, 8]. By definition, body-actuating systems take charge of the user's body and movements. Such systems especially weakens user agency when actuating the user's muscles contrary to the user's intent, or doing so preemptively before they intend to move [34, 35]. In one study [44], participants noted this agency loss after experiencing EMS changing their pose involuntarily: "it is weird that you lose against yourself," "sometimes the computer hand was faster than I," and "it was so remarkable to see my hand moving without my intention that I could not look away."

Enhancing user trust in body-actuating systems may mitigate the hesitance in system adoption. Despite reduced agency, if the user has confidence in the system's ability and reliability to keep them from harm's way, they will be more likely to take advantage of its benefits. Here we rely on the definition of trust prevalent in the scholarly literature as "the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party" (p.712) [51]. Trust is multidimensional, including capacity and moral trust [87]. These dimensions all affect user confidence in the system.

In this work, we examine how a system that controls agency may affect or foster user trust. This comes from prior observations that users might attribute the control they lost to other elements of the interaction, following an established distinction between "feeling of agency" (whether one feels to be in control of their actions, to which we sometimes refer as "user agency") and "judgment of agency" (whether one perceives themselves or others to have agency) [84]. Given reduced "feeling of agency," we examine whether "judgment of agency" affects user trust. In one study [45], EMS actuation manipulated participants' hands and arms to simulate how they should interact with a set of familiar and unfamiliar objects, e.g. an unusual avocado peeler. Participants used a variety of grammatical subjects to describe their experiences. Some attributed agency to the target object, e.g. "it [the avocado] didn't want me to cut it." Some attributed agency to the tool whose affordance was communicated, e.g. "The scooping tool told me to pull." Some attributed agency to the EMS system instead, e.g. (for a spray can) "the system shakes the can for me."

Through a human-subjects experiment, we examine how users attribute agency to the body-actuating system when the system includes voice explanations and anthropomorphism. By varying the presence of these two components, we further examine how we can build, maintain, and repair user trust in the system. Our research questions are:

- **RQ1:** Do voice explanations and anthropomorphism affect how people attribute agency to themselves and to the individual components of the body-actuating system?
- **RQ2:** Does change in agency attribution affect user trust in the system over time, including in instances of trust violation and subsequent attempts at repair?
- **RQ3:** Does change in agency attribution mediate the relationship between voice explanations or anthropomorphism and user trust in the system?

5.1 Project Plan

In a between-subject study, we will ask participants to complete a set of tasks while actuated by EMS. We focus on EMS as the body-actuating device, as it has promising applications [45, 46, 43, 15, 86] due to its lightweight nature compared to alternatives like exoskeletons. Participants will be randomly assigned to three conditions. In the baseline condition, participants complete tasks with EMS sometimes controlling their body movements in unexpected ways that are usually correct. In a second condition, participants will be similarly actuated by EMS, while a speaker nearby broadcast spoken explanations of the EMS actuation. In the last condition, participants will be similarly actuated by EMS, while a humanoid robot nearby will move in-sync and verbally explained the EMS actuation. With this setup, we will investigate how participants judge their own agency and the agency of each piece of technology, and how this judgment affects their trust in the body-actuating system.

CHAPTER 6
EXPECTED TIMELINE

Of this dissertation, work pertaining to automated services (Chapter 2), reinforcement learning (Chapter 3), and livestreamed performances (Chapter 4) has been completed. Remaining work pertains to the Body-Actuating Device Trust study (Chapter 5).

By Mar 10, 2023

- Acquire IRB approval.
- Complete experimental design and implementation.
- Pilot experiments.

By Jun 2, 2023

- Complete data collection.
- Complete data analysis.

By Mid Sept or Early Oct, 2023 (Depending on CHI/HRI Deadlines)

- Complete manuscript in preparation for CHI or HRI submission.
- Submit manuscript to CHI or HRI.

By Jan 19, 2024

- Submit initial full draft of dissertation to committee for feedback.

By Mar 22, 2024

- Submit final version of complete dissertation draft to committee for final feedback before the final submission date of May 1, 2024.

- Submit UChicago application to graduate.

By May 1, 2024

- Submit final copy of dissertation to UChicago.

REFERENCES

[1] Neziha Akalin and Amy Loutfi. Reinforcement learning approaches in social robotics, 2021.

[2] Sean Andrist, Micheline Ziadee, Halim Boukaram, Bilge Mutlu, and Majd Sakr. Effects of culture on the credibility of robot speech: A comparison between english and arabic. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '15, 2015.

[3] Koosha Araghi. Google docs has full 'track changes' word integration, 12 2014.

[4] Christian Arzate Cruz and Takeo Igarashi. *A Survey on Interactive Reinforcement Learning: Design Principles and Open Challenges*. 2020.

[5] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[6] Wilma A Bainbridge, Justin W Hart, Elizabeth S Kim, and Brian Scassellati. The benefits of interactions with physically present robots over video-displayed agents. *International Journal of Social Robotics*, 3:41–52, 2011.

[7] Andrew G Barto, Philip S Thomas, and Richard S Sutton. Some recent applications of reinforcement learning. In *Proceedings of the Eighteenth Yale Workshop on Adaptive and Learning Systems*, 2017.

[8] Bruno Berberian, Jean-Christophe Sarrazin, Patrick Le Blaye, and Patrick Haggard. Automation technology and sense of control: a window on human agency. *PloS one*, 7(3):e34075, 2012.

[9] Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L. Littman, and Blase Ur. How users interpret bugs in trigger-action programming. In *Proc. CHI*, 2019.

[10] Mason Bretan and Gil Weinberg. A survey of robotic musicianship. *Communications of the ACM*, 59(5):100–109, May 2016.

[11] Dražen Brščić, Hiroyuki Kidokoro, Yoshitaka Suehiro, and Takayuki Kanda. Escaping from children's abuse of social robots. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '15, 2015.

[12] Colleen M. Carpinella, Alisa B. Wyman, Michael A. Perez, and Steven J. Stroessner. The robotic social attributes scale (rosas): Development and validation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, page 254–262, New York, NY, USA, 2017. Association for Computing Machinery.

[13] Ryan Chard, Kyle Chard, Jason Alt, Dilworth Y. Parkinson, Steve Tuecke, and Ian Foster. Ripple: Home automation for research data management. In *Proc. ICDCSW*, 2017.

[14] Ryan Chard, Rafael Vescovi, Ming Du, Hanyu Li, Kyle Chard, Steve Tuecke, Narayanan Kasthuri, and Ian Foster. High-throughput neuroanatomy and trigger-action programming: A case study in research automation. In *Proc. AI-Science*, 2018.

[15] Yuxin Chen, Zhuolin Yang, Ruben Abbou, Pedro Lopes, Ben Y. Zhao, and Haitao Zheng. User authentication via electrical muscle stimulation. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Sven Coppers, Davy Vanacken, and Kris Luyten. Fortniot: Intelligible predictions to improve user understanding of smart home behavior. *PACM IMWUT*, 4(4), 2020.

[17] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. Empowering end users in debugging trigger-action rules. In *Proc. CHI*, 2019.

[18] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. My iot puzzle: Debugging if-then rules through the jigsaw metaphor. In *Proc. IS-EUD*, 2019.

[19] Joe Davison. distilbert-base-uncased-go-emotions-student. `https://huggingface.co/joeddav/distilbert-base-uncased-go-emotions-student`. Accessed: 2021-06-23.

[20] Ben Francis. Introducing mozilla webthings, April 2019.

[21] Leslie J. Francis, Laurence B. Brown, and Ronald Philipchalk. The development of an abbreviated form of the revised eysenck personality questionnaire (epqr-a): Its use among students in england, canada, the u.s.a. and australia. *Personality and Individual Differences*, 13(4):443–449, 1992.

[22] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. Personalization of context-dependent applications through trigger-action rules. *TOCHI*, 2017.

[23] Baldwin Giang. Works - Baldwin Giang: MUSICBOT1975 (2021). `https://baldwingiang.com/works`. Accessed: 2021-11-23.

[24] Xiaochi Gu, Yifei Zhang, Weize Sun, Yuanzhe Bian, Dao Zhou, and Per Ola Kristensson. Dexmo: An inexpensive and lightweight mechanical exoskeleton for motion capture and force feedback in vr. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 1991–1995, New York, NY, USA, 2016. Association for Computing Machinery.

[25] Guy Hoffman. On stage: Robots as performers. In *Robotics: Science and Systems 2011 Workshop on Human-Robot Interaction*, 2011.

[26] Guy Hoffman. Dumb robots, smart phones: A case study of music listening companionship. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 358–363, 2012.

[27] Guy Hoffman, Shira Bauman, and Keinan Vanunu. Robotic experience companionship in music listening and video watching. *Personal Ubiquitous Comput.*, 20(1):51–63, February 2016.

[28] Guy Hoffman and Gil Weinberg. Gesture-based human-robot jazz improvisation. In *2010 IEEE International Conference on Robotics and Automation*, pages 582–587, 2010.

[29] Guy Hoffman and Gil Weinberg. Interactive improvisation with a robotic marimba player. *Autonomous Robots*, 31:133–153, 10 2011.

[30] Justin Huang and Maya Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proc. UbiComp*, 2015.

[31] IEEE and The Open Group. diff, 2018.

[32] IFTTT. Ifttt helps your apps and devices work together, 2019.

[33] Myounghoon Jeon, Maryam Fakhr Hosseini, Jaclyn Barnes, Zackery Duford, Ruimin Zhang, Joseph Ryan, and Eric Vasey. Making live theatre with multiple robots as actors: Bringing robots to rural schools to promote steam education for underserved students. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, HRI '16, page 445–446. IEEE Press, 2016.

[34] Shunichi Kasahara, Jun Nishida, and Pedro Lopes. Preemptive action: Accelerating human reaction using electrical muscle stimulation without compromising agency. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Shunichi Kasahara, Kazuma Takada, Jun Nishida, Kazuhisa Shibata, Shinsuke Shimojo, and Pedro Lopes. Preserving agency during electrical muscle stimulation training speeds up reaction time directly after removing ems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[36] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 2013.

[37] Min Kyung Lee, Jodi Forlizzi, Sara Kiesler, Paul Rybski, John Antanitis, and Sarun Savetsila. Personalization in hri: A longitudinal field experiment. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, 2012.

[38] Nicola Leonardi, Marco Manca, Fabio Paternò, and Carmen Santoro. Trigger-action programming for personalising humanoid robot behaviour. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, 2019.

[39] Daniel Leyzberg, Samuel Spaulding, and Brian Scassellati. Personalizing robot tutors to individuals' learning differences. In *Proceedings of the 2014 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '14, 2014.

[40] Abner Li. Google docs rolling out dedicated 'compare documents' tool, June 2019.

[41] Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Börje F. Karlsson, Dongmei Zhang, and Feng Zhao. Systematically debugging iot control system correctness for building automation. In *Proc. BuildSys*, 2016.

[42] Hannah Limerick, David Coyle, and James W. Moore. The experience of agency in human-computer interactions: a review. *Frontiers in Human Neuroscience*, 8, 2014.

[43] Pedro Lopes and Patrick Baudisch. Interactive systems based on electrical muscle stimulation. *Computer*, 50(10):28–35, 2017.

[44] Pedro Lopes, Alexandra Ion, Willi Mueller, Daniel Hoffmann, Patrik Jonell, and Patrick Baudisch. Proprioceptive interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 939–948, New York, NY, USA, 2015. Association for Computing Machinery.

[45] Pedro Lopes, Patrik Jonell, and Patrick Baudisch. Affordance++: Allowing objects to communicate dynamic use. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 2515–2524, New York, NY, USA, 2015. Association for Computing Machinery.

[46] Pedro Lopes, Sijing You, Lung-Pan Cheng, Sebastian Marwecki, and Patrick Baudisch. Providing haptics to walls & heavy objects in virtual reality by means of electrical muscle stimulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 1471–1482, New York, NY, USA, 2017. Association for Computing Machinery.

[47] Pedro Lopes, Sijing You, Alexandra Ion, and Patrick Baudisch. Adding force feedback to mixed reality experiences and games using electrical muscle stimulation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

[48] Chance Lytle, Parker Ramsey, Joey Yeo, Trace Dressen, Dong hyun Kang, Brenda Bakker Harger, and Jessica Hammer. Toward live streamed improvisational game experiences. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '20, page 148–159, New York, NY, USA, 2020. Association for Computing Machinery.

[49] Chance Lytle, Parker Ramsey, Joey Yeo, Trace Dressen, Dong hyun Kang, Joseph Seering, Brenda Bakker Harger, and Jessica Hammer. Social design for complex participatory livestream activities. In *The 2020 CHI Conference on Human Factors in Computing Systems Workshop on Be Part Of It: Spectator Experience in Gaming and eSports*, 2020.

[50] Marco Manca, Fabio, Paternò, Carmen Santoro, and Luca Corcella. Supporting end-user debugging of trigger-action rules for iot applications. *IJHCS*, 2019.

[51] Roger C. Mayer, James H. Davis, and F. David Schoorman. An integrative model of organizational trust. *The Academy of Management Review*, 20(3):709–734, 1995.

[52] Edward J. McCluskey. *Logic Design Principles with Emphasis on Testable Semicustom Circuits*. Prentice-Hall, Inc., USA, 1986.

[53] Sarah Mennicken, David Kim, and Elaine May Huang. Integrating the smart home into the digital calendar. In *Proc. CHI*, 2016.

[54] Aaron Meurer et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 2017.

[55] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. An empirical characterization of ifttt: Ecosystem, usage, and performance. In *Proc. IMC*, 2017.

[56] Joseph E Michaelis and Bilge Mutlu. Reading socially: Transforming the in-home reading experience with a learning-companion robot. *Science Robotics*, 3(21):eaat5999, 2018.

[57] Claire Mikalauskas, Tiffany Wun, Kevin Ta, Joshua Horacsek, and Lora Oehlberg. Improvising with an audience-controlled robot performer. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, page 657–666, New York, NY, USA, 2018. Association for Computing Machinery.

[58] Kazuki Mizumaru, Satoru Satake, Takayuki Kanda, and Tetsuo Ono. Stop doing it! approaching strategy for a robot to admonish pedestrians. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 449–457. IEEE, 2019.

[59] James W Moore. What is the sense of agency and why does it matter? *Frontiers in psychology*, 7:1272, 2016.

[60] Mozilla. Announcing "project things" - an open framework for connecting your devices to the web., Feb 2018.

[61] MSFTMan. Create a flow in microsoft flow, May 2019.

[62] Jun Nishida, Shunichi Kasahara, and Kenji Suzuki. Wired muscle: Generating faster kinesthetic reaction by inter-personally connecting muscles. In *ACM SIGGRAPH 2017 Emerging Technologies*, SIGGRAPH '17, New York, NY, USA, 2017. Association for Computing Machinery.

[63] Jun Nishida, Yudai Tanaka, Romain Nith, and Pedro Lopes. Digitusync: A dual-user passive exoskeleton glove that adaptively shares hand gestures. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery.

[64] Tatsuya Nomura, Takayuki Uratani, Takayuki Kanda, Kazutaka Matsumoto, Hiroyuki Kidokoro, Yoshitaka Suehiro, and Sachie Yamada. Why do children abuse robots? In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*, HRI'15 Extended Abstracts, 2015.

[65] D. A. Norman. The 'problem' with automation: Inappropriate feedback and interaction, not 'over-automation'. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 327(1241):585–593, 1990.

[66] Mark Otto. Introducing split diffs, Sept 2014.

[67] Stefan Palan and Christian Schitter. Prolific.ac - A subject pool for online experiments. *JBEF*, 2018.

[68] Mitali Palekar, Earlence Fernandez, and Franziska Roesner. Analysis of the susceptibility of smart home programming interfaces to end user error. *Proc. SafeThings*, 2019.

[69] Caroline Pantofaru, Leila Takayama, Tully Foote, and Bianca Soto. Exploring the role of robots in home organization. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 327–334, 2012.

[70] Eyal Peer, Laura Brandimarte, Sonam Samat, and Alessandro Acquisti. Beyond the turk: Alternative platforms for crowdsourcing behavioral research. *JESP*, 2017.

[71] Andrea Piscitello, Alessandro A. Nacci, Vincenzo Rana, Marco D. Santambrogio, and Donatella Sciuto. Ruleset minimization in multi-tenant smart buildings. In *Proc. CSE and EUC and DCABES*, 2016.

[72] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, 2009.

[73] Justin Pot. What is version history and how to use it in google docs?, April 2019.

[74] Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. IFTTT vs. zapier: A comparative study of trigger-action programming frameworks. *CoRR*, 2017.

[75] Aditi Ramachandran, Alexandru Litoiu, and Brian Scassellati. Shaping productive help-seeking behavior during robot-child tutoring interactions. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 247–254. IEEE, 2016.

[76] P. Salvini, G. Ciaravella, W. Yu, G. Ferri, A. Manzi, B. Mazzolai, C. Laschi, S.R. Oh, and P. Dario. How safe are service robots in urban environments? bullying a robot. In *19th International Symposium in Robot and Human Interactive Communication*, 2010.

[77] Allison Sauppé and Bilge Mutlu. Robot deictics: How gesture and context shape referential communication. In *Proceedings of the 2014 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '14, 2014.

[78] Joseph Seering, Juan Pablo Flores, Saiph Savage, and Jessica Hammer. The social roles of bots: Evaluating impact of bots on discussions in online communities. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[79] Joseph Seering, Saiph Savage, Michael Eagle, Joshua Churchin, Rachel Moeller, Jeffrey P. Bigham, and Jessica Hammer. Audience participation games: Blurring the line between player and spectator. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, DIS '17, page 429–440, New York, NY, USA, 2017. Association for Computing Machinery.

[80] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction.* Pearson, 2016.

[81] Sarah Strohkorb Sebo, Margaret Traeger, Malte Jung, and Brian Scassellati. The ripple effects of vulnerability: The effects of a robot's vulnerable behavior on trust in human-robot teams. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '18, page 178–186, New York, NY, USA, 2018. Association for Computing Machinery.

[82] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes. In *Proc. WWW*, 2017.

[83] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* 1998.

[84] Matthis Synofzik, Gottfried Vosgerau, and Albert Newen. Beyond the comparator model: A multifactorial two-step account of agency. *Consciousness and Cognition*, 17(1):219–239, 2008.

[85] Emi Tamaki, Takashi Miyaki, and Jun Rekimoto. Possessedhand: Techniques for controlling human hands using electrical muscles stimuli. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 543–552, New York, NY, USA, 2011. Association for Computing Machinery.

[86] Yudai Tanaka, Jun Nishida, and Pedro Lopes. Electrical head actuation: Enabling interactive systems to directly manipulate head orientation. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[87] Daniel Ullman and Bertram F. Malle. What does it mean to trust a robot? steps toward a multidimensional measure of trust. In *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '18, page 263–264, New York, NY, USA, 2018. Association for Computing Machinery.

[88] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. Practical trigger-action programming in the smart home. In *Proc. CHI*, 2014.

[89] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proc. CHI*, 2016.

[90] John Vilk and Naomi T. Fitter. Comedians in cafes getting data: Evaluating timing and adaptivity in real-world robot comedy performance. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '20, page 223–231, New York, NY, USA, 2020. Association for Computing Machinery.

[91] Lin Wang, Pei-Luen Patrick Rau, Vanessa Evers, Benjamin Krisper Robinson, and Pamela Hinds. When in rome: The role of culture & context in adherence to robot recommendations. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*, HRI '10, 2010.

[92] Svetlana Yarosh and Pamela Zave. Locked or not?: Mental models of iot feature interaction. In *Proc. CHI*, 2017.

[93] Zapier. How zapier works, 2019.

[94] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. Autotap: Synthesizing and repairing trigger-action programs using ltl properties. In *Proc. ICSE*, 2019.

[95] Lefan Zhang, Weijia He, Olivia Morkved, Valerie Zhao, Michael L. Littman, Shan Lu, and Blase Ur. Trace2tap: Synthesizing trigger-action programs from traces of behavior. *PACM IMWUT*, 4(3), 2020.

[96] Valerie Zhao, Lefan Zhang, Bo Wang, Michael L. Littman, Shan Lu, and Blase Ur. Supplementary materials for understanding trigger-action programs through novel visualizations of program differences, 2021.

[97] A.B. Zoss, H. Kazerooni, and A. Chu. Biomechanical design of the berkeley lower extremity exoskeleton (bleex). *IEEE/ASME Transactions on Mechatronics*, 11(2):128–138, 2006.