A Formal Method for Including the Probability of Erroneous Human Task Behavior in System Analyses

Matthew L. Bolton[a,*], Xi Zheng[a], Eunsuk Kang[b]

[a] *University at Buffalo, State University of New York, Department of Industrial and Systems Engineering, Buffalo, NY, USA*
[b] *Institute for Software Research, School of Computer Science, Carnegie Mellon University, NY, USA*

Abstract

Formal methods have been making inroads into the engineering of human-automation interaction (HAI) by allowing engineers to use mathematical proofs to determine whether normative or unanticipated erroneous human behavior can ever cause problems. However, these approaches are limited because they do not give engineers a way to assess the relative likelihood of different outcomes. In this work, we address this shortcoming by defining a new approach that combines formal approaches with human reliability analysis and probabilistic and statistical model checking. This approach ultimately allows analysts to compute the probability of different outcomes occurring in reactive HAI systems. We describe how this method was realized, assess its scalability, and demonstrate its capabilities with an automated teller machine example. We ultimately discuss our results and describe directions of future research.

Keywords: Human error, formal methods, model checking, probabilistic modeling, human reliability.

1. Introduction

Human error is regularly cited as a source of failure and performance issues in modern systems. It has contributed to more than 1,000,000 injuries and between 44,000 and 98,000 deaths annually in medicine (Kohn et al., 2000); roughly 75% of all accidents in general aviation and 50% in commercial aviation (Kebabjian, 2018; Kenny, 2015); one third of unmanned aerial system (UAS) accidents (Manning et al., 2004); 90% of automobile crashes (NHTSA, 2008); and high profile disasters like the accident at Three Mile Island (Le Bot, 2004). Humans are often blamed for failures associated with human error. However, the modern perspective holds that errors are the result of shortcomings in system design and thus not solely the fault of human operators, if they are the fault of the human at all. Unfortunately, the complexity of human-automation interaction (HAI) systems and the inherent concurrency that exists in HAI can make it extremely difficult to predict when and how erroneous human behaviors can cause problems. For this reason, a growing body of research has been investigating how rigorous analyses from formal methods can be used to evaluate and design HAI (Bolton, 2017a; Bolton et al., 2013; Weyers et al., 2017). In particular, techniques have been discovered that enable an analyst to pair models of human tasks (a normative description of the behavior humans use to achieve goals when interacting with a system) with models of system functionality. Verification techniques are then used to determine if normative or erroneous human behavior (based on systematic deviations from normative tasks) can result in safety violations (Aït-Ameur and Baron, 2006; Barbosa et al., 2011; Bastide and Basnyat, 2007; Bolton and Bass, 2017; Fields, 2001; Paternò and Santoro, 2001).

These techniques are powerful and well-suited to discovering design flaws in HAIs, especially those involving potentially unanticipated erroneous human behaviors, and thus suggesting interventions that improve system reliability. However, they do have limitations. In particular, they do not account for the relative likelihood of different erroneous behaviors. This situation can make it difficult for analysts to prioritize how to address discovered problems. This research seeks to address this deficiency by extending concepts from formal, task-based, verification methods with concepts from human reliability analysis (HRA) and probabilistic / statistical model checking (methods for formally verifying stochastic systems). In what follows, we cover the background necessary for understanding our approach, we describe our method, we analyze a proof of concept example [an automated teller machine (ATM)] to check that our method is providing valid results. Finally, we discuss our developments and discuss directions for future work.

2. Background

2.1. Formal Methods

Formal methods is a sub-domain of computer science concerned with mathematically modeling systems with precise languages, specifying desirable system properties, and verifying (proving) whether the properties hold with the system. Model checking (Clarke et al., 1999) is computer software that performs formal verification automatically, where efficient data structures and algorithms are used to search through a system model's state space to exhaustively determine if specification properties hold.

*Corresponding author
Email address: mbolton@buffalo.edu (Matthew L. Bolton)

The majority of formal methods are non-stochastic. Emerging techniques such as probabilistic and statistical model checking enable analysts to account for probabilities in formal verification (Kwiatkowska et al., 2007). In these, stochastic models (such as Markov chains) are used to describe system behavior and the specification properties either describe desirable system properties (which can include probabilities) or request that a probability be computed. Verification then proves whether the specification property holds for the entire model or computes the probability requested in the property. Statistical model checking only differs from probabilistic checking in that, instead of verifying a property against the entire system model, statistical checking checks the property against a number of samples/traces through the model. This produces approximate results. For situations where the property is computing a probability, this means that statistical checking produces a probability value and an accurate confidence interval (CI) for it.

2.2. Formal and Task-analytic Methods

Formal methods (and especially model checking) are adept at discovering unexpected interactions that can cause system failures. Because of this, a growing literature has been investigating how formal methods can be used to engineer HAI. Comprehensive reviews can be found in (Bolton, 2017a; Bolton et al., 2013; Weyers et al., 2017). Due to topicality, what follows focuses on formal methods that have used task analytic methods and as well as those that have attempted to model human error stochastically.

Task analysis is a systematic process human factors engineers use to describe how humans normatively achieve goals with a system as a task model (Schraagen et al., 2000). Task models can be interpreted formally and thus included in larger formal analyses. This means that formal verification can prove whether human behavior captured in task models can cause performance and safety issues and be used to investigate solutions to problems. Task models can be formally modeled manually (Basnyat et al., 2007; Gunter et al., 2009). However, it is more common to automatically translate tasks represented in their own notations into a formalism where other system behavior is described (Aït-Ameur and Baron, 2006; Bolton and Bass, 2010a; Bolton et al., 2011; Fields, 2001; Palanque et al., 1996; Paternò and Santoro, 2001).

Additionally, researchers have discovered a number of approaches for injecting or generating human error into previously normative task models so that the impact of both anticipated and potentially unanticipated erroneous behaviors can be accounted for in verifications. These approaches either use mutation patterns (which are manually applied to task models by analysts) or different theories or taxonomies of human error to automatically include deviations from task in formal representations (Barbosa et al., 2011; Bastide and Basnyat, 2007; Bolton, 2015; Bolton and Bass, 2011, 2013a,b; Bolton et al., 2012, 2019; Fields, 2001; Pan and Bolton, 2018).

Because of its support for including human behavior in formal verification analyses, especially as it relates to the automated generation of potentially unanticipated human errors, we use the enhanced operator function model (EOFM) for task modeling in this research. We discuss EOFM next.

2.2.1. The Enhanced Operator Function Model

EOFM (Bolton et al., 2011) is an XML-based task modeling formalism where human behavior is captured as an input/output model. Inputs are system elements external to the human (e.g., interface display information and observable environment elements). Outputs are human actions. The operators' tasks describe how human actions occur based on input and local variables, where local variables represent the human's perceptual or cognitive state.

Tasks are a hierarchy of activities and actions (acts) starting with a root activity. Any activity (which represents a goal-directed complex behavior) can decomposes into sub-activities and, ultimately, atomic actions (concrete cognitive or observable behaviors that are not further broken down). Strategic knowledge (Boolean expressions) can be specified for each activity using input and local variables. An activity can have three different conditions asserting what must be true for the activity to start executing (*Precondition*); repeat (*RepeatCondition*), or complete (*CompletionCondition*).

A decomposition has an operator that constrains how many of the acts in that decomposition can execute as well as the (ordinal) temporal relationships between them. EOFM supports ten such operators, those topical to the paper are shown in Table 1.

Atomic human actions or internal actions (representing cognitive or perceptual behaviors) occur at the bottom of every task. Human actions can have one of three behaviors: *AutoReset* actions occur as a single event; *Toggle* actions switch between occurring and not occurring; and *SetValue* actions commit some value to the other parts of the system. Internal actions allow cognitive behaviors (such as remembering something) to be modeled by assigning values to local variable.

To enable unambiguous interpretation of EOFM behavior, EOFMs have formal semantics (Bolton et al., 2011, 2016). In this, every act is interpreted as a state machine (Fig. 1) with three states: *Ready*, *Executing*, and *Done*. All acts start in *Ready*. They transition between states based on whether the Boolean conditions on the labeled transitions (Fig. 1) are true. When

Table 1: Decomposition Operators (Bolton and Bass, 2011)

Operator	Description
optor_seq	Zero or more of the activities or actions in the decomposition must execute in any order, one at a time.
optor_par	Zero or more of the activities or actions in the decomposition must execute in any order and can execute in parallel.
or_seq	One or more of the activities or actions in the decomposition must execute in any order one at a time.
or_par	One or more of the activities or actions in the decomposition must execute in any order and can execute in parallel.
and_seq	All activities or actions in the decomposition must execute in any order, one at a time.
and_par	All activities or actions in the decomposition must execute in any order and can execute in parallel.
xor	Exactly one activity or action in the decomposition executes.
ord	All activities or actions must execute in the order (left to right) they appear in the decomposition.

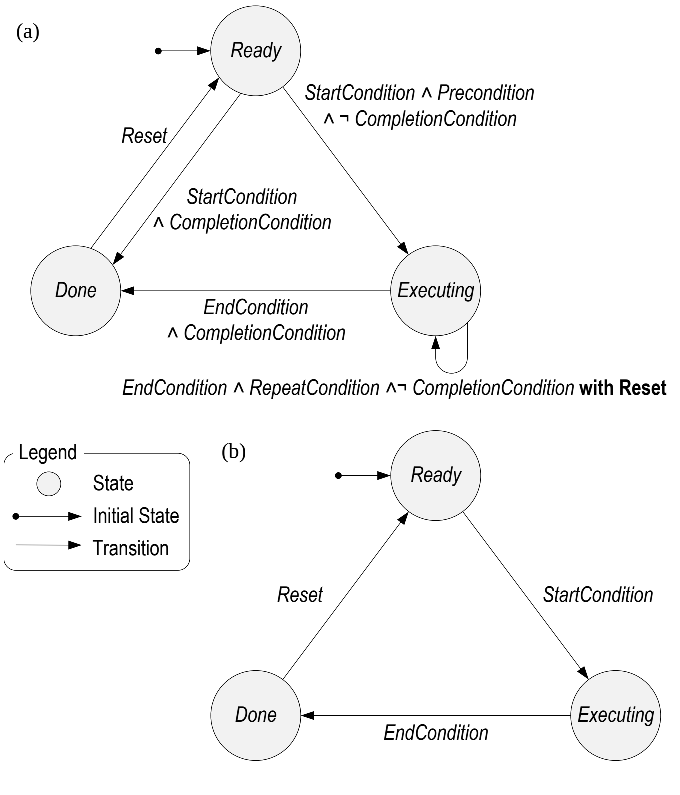

(a)
Ready
StartCondition ∧ Precondition ∧ ¬ CompletionCondition
Reset
StartCondition ∧ CompletionCondition
Done
Executing
EndCondition ∧ CompletionCondition
EndCondition ∧ RepeatCondition ∧¬ CompletionCondition with Reset
Legend
State
Initial State
Transition
(b)
Ready
Reset
StartCondition
Done
Executing
EndCondition

Figure 1: EOFM transition semantics. (a) Transition semantics for an EOFM activity. (b) Transition semantics for an EOFM action.

an action is *Executing*, this corresponds to an output variable associated with the action being set to the appropriate value. For *AutoReset* actions this is a Boolean variable that is set to true. For *Toggle*, the variable's Boolean variable is set to the negation of its current value. For *SetValue* behavior, the action's output variable is set to the value assigned in the model. The same occurs when the action is a local variable assignment.

Any existent activity strategic knowledge conditions (*Preconditions*, *RepeatConditions*, and *CompletionConditions*) partially describe when transitions can occur. However, there are three additional implicit conditions (based on the activity's or action's task position) that also impact transitions.

The *StartCondition* indicates if an act can start executing based on the states of its parent and siblings (acts in the same decomposition) as well as the parent's decomposition operator. Mathematically, a *StartCondition* has the generic form:

$$\begin{aligned} StartCondition: \quad & parent.state = Executing \\ & \wedge \bigwedge_{\forall siblings\ s} (s.state \neq Executing)\,. \end{aligned} \tag{1}$$

If the parent's decomposition allows for parallelism (the operator ends with _par), the second conjunct is eliminated. If the parent has an ord decomposition, to enforce order, the second conjunct requires that the previous sibling in the decomposition be done: $(prev\_sibling.state = Done)$. If the parent has an xor decomposition, the second conjunct is modified so that no other sibling can execute after one has finished: $\bigwedge_{\forall siblings\ s} (s.state = Ready)$. An activity without a parent (a top-level activity) will eliminate the first conjunct. Top-level activities treat each other as siblings in an and_seq formulation for the second conjunct.

The *EndCondition* indicates if an act can stop executing based on the execution state of its children:[1]

$$\begin{aligned} EndCondition: \quad & \bigodot_{\forall subacts\ c} (c.state = Done) \\ & \wedge \bigwedge_{\forall subacts\ c} (c.state \neq Executing)\,. \end{aligned} \tag{2}$$

The first conjunct requires that the execution states of the activity's children satisfy the activity's decomposition operator. Here, $\bigodot$ is generic and substituted with $\bigwedge$ if the activity has an and_seq, and_par, or ord decomposition. It is substituted with $\bigvee$ for activities with or_seq, or_par, or xor decompositions. Because optor_seq and optor_par do not impose restrictions on the number of children that can execute, the first conjunct is eliminated for either decomposition. The second conjunct always specifies that no children are *Executing*.

The *Reset* describes when an activity can return to *Ready*. It occurs in two situations. First, when an activity repeats (a normative *Executing*-to-*Executing* transition), every descendant act *Resets*. Second, any root activity automatically *Resets* from *Done*. When this occurs, all descendants *Reset*.

The formal semantics of EOFM were used to create automated translation software that converted EOFM XML into the input language of the Symbolic Analysis Laboratory (SAL), a suite of non-probabilistic model checkers (Bolton and Bass, 2017; Bolton et al., 2011, 2016). This enables EOFM behavior to be used as part of larger formal system analyses. As a result, EOFM analyses have been used to assess the impact of normative human behavior on a number of automotive, aerospace, and medical systems (Bolton and Bass, 2009a,b, 2010a, 2012). EOFM has also been used to model and predict the impact of erroneous human behavior.

2.2.2. EOFM and Erroneous Human Behavior

EOFM has been used to automatically generate human errors using multiple theories (Bolton and Bass, 2010b, 2011, 2013a,b, 2017; Bolton et al., 2012). This culminated in EOFM serving as the basis for the task-based taxonomy of erroneous human behavior (Bolton, 2017b) and the associated error generation approach (Bolton et al., 2019). The task-based taxonomy unifies the phenomenological (what) and genotypical (why) perspectives by classifying human error based on where they deviate from tasks. By knowing where a deviation occurs in a task model (how the task's formal semantics were violated), one understands how the error will observably manifest (the phenotype; Hollnagel 1993) and why the error occurred (the strategic knowledge or inherent condition the human improperly attended to; the genotype of the slip; Reason 1990). In fact, the task-based taxonomy has been shown to be complete with respect to both the leading phenomenological (Hollnagel, 1993) and genomenological (Reason, 1990) taxonomies (Bolton, 2017b).

The task-based taxonomy has a hierarchical classification that goes beyond what is relevant to this discussion (Bolton, 2017b). What is topical is that it distinguishes between semantic

[1]Because actions have no children, action *EndConditions* default to true.

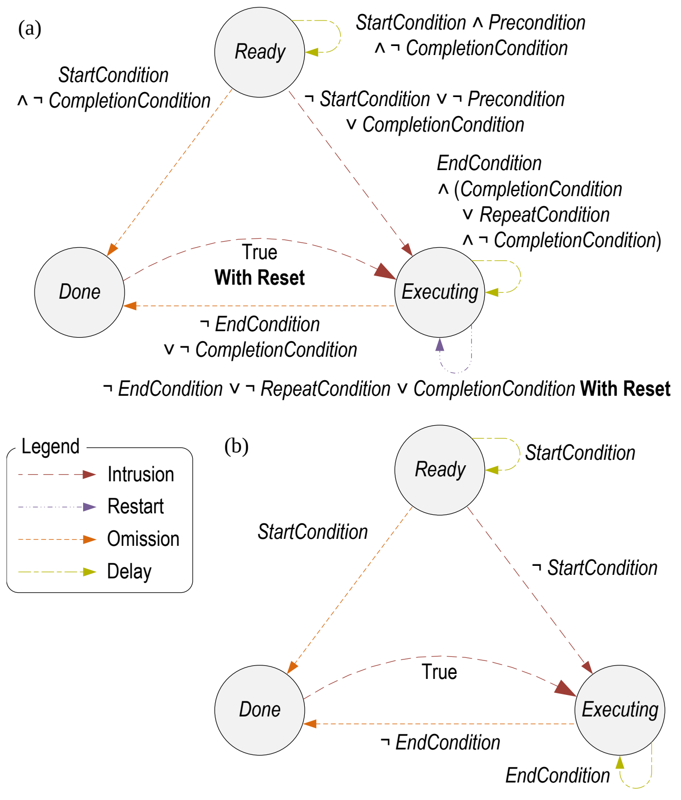

(a)
Ready
StartCondition ∧ Precondition ∧ ¬ CompletionCondition
StartCondition ∧ ¬ CompletionCondition
¬ StartCondition ∨ ¬ Precondition ∨ CompletionCondition
EndCondition ∧ (CompletionCondition ∨ RepeatCondition ∧ ¬ CompletionCondition)
Done
True
With Reset
Executing
¬ EndCondition ∨ ¬ CompletionCondition
¬ EndCondition ∨ ¬ RepeatCondition ∨ CompletionCondition With Reset
Legend
Intrusion
Restart
Omission
Delay
(b)
Ready
StartCondition
StartCondition
¬ StartCondition
Done
True
Executing
¬ EndCondition
EndCondition

Figure 2: Task-based taxonomy of human error transition-based error modes.

violations that are transition-based (transitions that violate what is in Fig. 1) and assignment-based (incorrect behavior assignment to an action variable during action execution for *SetValue* or local variable actions). Transition-based error modes describe erroneous transitions between act execution states (see Fig. 2). An intrusion happens when an act executes (transitions to *Executing*) when it should not. An omission manifests when an activity finishes (transitions to *Done*) when it should not. A restart occurs when an activity (but not an action) incorrectly restarts: the activity resets and starts executing anew. Finally, a delay happens when an act does not transition out of a state when it should. In the execution-based erroneous behaviors, there can be both value and target substitutions (for *SetValue* actions) and misremembrances (for local variable assignments) (Table 2). In a value substitution or misremembrance, an incorrect value is assigned to the correct target variable (the action's output or local variable). In a target error, the correct value is assigned to the wrong target (the wrong output or local variable).

This taxonomy was the basis for a method (Bolton et al., 2019) to automatically generate erroneous behavior in model checking analyses to potentially discover (and fix) human errors that could cause system failures. Specifically, this method would modify the EOFM-to-SAL translator so that all erroneous transitions (Fig. 2) and assignments (Table 2) could exist alongside their normative counterparts (Fig. 1). A maximum (*Max*) on the total number of errors included was used to keep models from becoming completely unbounded. The net effect of this was that analyses accounted for all of the different ways that *Max* errors could occur when evaluating system safety.

This approach is powerful and particularly good at discovering HAI design flaws that could interact with unanticipated erroneous human behaviors. However, it does have limits. In particular, it does not account for the relative likelihood of different erroneous behaviors. This means that analyses may discover errors that are extremely unlikely to occur. Thus, with no means of ranking the relative probability of different errors, it can be difficult for analysts to prioritize how to address discovered problems. None of the formal verification work that has used task analytic or cognitive human behavior has accounted for the probabilities of different errors. To address this shortcoming, we plan to use concepts from HRA.

Table 2: Action-level Erroneous Behavior Types

Assignment	Erroneous Behavior Type
$CorrectAction := IncorrectValue$	Action Value Substitution
$IncorrectAction := CorrectValue$	Action Target Substitution
$CorrectVariable := IncorrectValue$	Value Misremembrance
$IncorrectVariable := CorrectValue$	Target Misremembrance

2.3. *HRA and CREAM*

HRAs are used to predict human error rates. There are many different HRAs (Bell and Holroyd, 2009). These generally fall into two categories: those based on probabilistic risk assessment (the first generation) and those based on cognition (the second generation) (Di Pasquale et al., 2013). Third generation techniques (which are discussed subsequently) exist, but use the theoretical foundation of first- and second-generation methods.

First-generation methods like the Technique for Human Error Rate Prediction (THERP) (Swain, 1987) and the Human Error Assessment and Reduction Technique (HEART) (Williams, 1986) (among others) are useful. However, they generally treat human errors the same as equipment failures. This limits their applicability because they do not account for the effect of context and organizational factors (Di Pasquale et al., 2013; Hollnagel, 1998a). Second-generation HRAs build off and improve on these by considering the interactions inherent in complex systems (between humans, processes, organization, and the environment) based on how they affect human cognition (Fujita and Hollnagel, 2004; Hollnagel, 1998a; Reer, 2008). CREAM is largely the preferred HRA by human factors engineers because it is rooted in a well-established cognitive model and can thus explain why errors occur (Blom et al., 2001; Di Pasquale et al., 2013; Hollnagel, 1998b; Stanton et al., 2001; Worm, 2001). Beyond this, CREAM has been well validated over the years through its successful use in nuclear power plants (Hollnagel et al., 1999), manufacturing (Geng et al., 2015), radiation therapy (Castiglia et al., 2008), hospitals (Rantanen et al., 2012), and many other critical domains (Chen et al., 2019; Hollnagel, 1998a; Rashed, 2016; Yang et al., 2013; Zhang et al., 2019; Zheng et al., 2020a,b, 2017).

In CREAM (Hollnagel, 1998a), analysts identify the major tasks for working with a given system. The analysts then perform subjective assessments with subject matter experts. First, common performance conditions (or CPCs) are assessed for each task to understand how the work environment impacts the

Table 3: CREAM's Cognitive Function Failures (Hollnagel, 1998a)

Cognitive Function	Cognitive Function Failure
Observation	Wrong Object Observed Wrong Identification Observation Not Made
Interpretation	Faulty Diagnosis Decision Error Delayed Interpretation
Planning	Priority Error Inadequate Plan
Execution	Action of Wrong Type Action of Wrong Time Action on Wrong Object Action Out of Sequence Missed Action

performance of that task. This means determining if each of the nine CPCs (quality of the organization, work conditions, human-machine support, procedures, simultaneous goals, time availability, time of day, work experience, and team collaboration) improves, reduces, or does not impact human performance. By synthesizing these ratings (counting the number that improve and reduce performance, and adjusting for dependencies), an analyst determines if the person is operating at one of the four Contextual Control Model (COCOM) modes (strategic, tactical, opportunistic, and scrambled; listed in decreasing levels of human reliability). In the basic version of CREAM, these control modes map to a range of probabilities of error occurring.

Point estimates can be achieved with two versions of extended CREAM. In these, an analyst identifies each task's cognitive function: observation – observing information in the environment; interpretation – understanding observed information; planning – setting a course of action; and execution – performing the planned actions. Next, the analyst identifies the function's most likely cognitive function failure (Table 3). Each failure has an associated nominal probability, originally identified by analyzing a large database of human performance information (Hollnagel, 1998a). In the first approach to extended CREAM, the nominal probability is scaled to account for the impact of CPCs by multiplying it by a scaling factor associated with the assessed control mode. In the second variant, the nominal probability is scaled based on the impact specific CPC levels have on the different cognitive functions.

CREAM has been used successfully in a number of safety critical environments. However, it does have some problems. Bedford et al. (2013) noted large error ranges and inconsistencies between predictions made with the three versions of CREAM. To address this, they created an improved version that reconciled the basic and extended methods (itself a refinement of previous attempts to improve CREAM predictions; Fujita and Hollnagel 2004; He et al. 2008). First, the method accounts for the impact of each CPC on overall performance in a single integer, thus avoiding the collapse of ratings into course control mode designations. This is computed as the total number of CPCs that

Table 4: Parameter Values Used for Calculating Probabilities of Human Error in the Revised Version of CREAM (Bedford et al., 2013)

	Cognitive Function			
Parameter	Observation	Interpretation	Planning	Execution
C	-2.0775	-1.3495	-2.0000	-2.4120
a	0.0055	0.0041	0.0052	0.0065
b	-0.2458	-0.2046	-0.2828	-0.2860
c	0.2840	0.2244	0.4019	0.4079

improve human performance and the number that reduce it:

$$\begin{aligned} CPC_{Sum} = \; & \left(\Sigma_{cpc \in \mathrm{CPCs}} \begin{cases} 1, & \text{if } cpc \text{ improves} \\ 0, & \text{otherwise} \end{cases} \right) \\ & - \left(\Sigma_{cpc \in \mathrm{CPCs}} \begin{cases} 1, & \text{if } cpc \text{ reduces} \\ 0, & \text{otherwise} \end{cases} \right). \end{aligned} \tag{3}$$

Second, using the same data that was the basis for CREAM's predictions (Hollnagel, 1998a), Bedford et al. (2013) fitted a regression model for predicting the probability of human error

$$P_{\mathrm{HumanError}} = 10^{\left(C + a \cdot CPC_{Sum}^2 + b \cdot CPC_{Sum} + c\right)} \tag{4}$$

where C is the $\log_{10}$ of the nominal probability of error for the cognitive function and a, b, and c are the regression coefficients for that function. See Table 4 for the computed values. Thus, by computing the CPC_{Sum} with (3) and using the result in (4) with the parameters from Table 4, accurate predictions of human error probabilities for a given task can be computed.

While CREAM (and its variants) provides a foundation for predicting error probabilities, it is a very manual process that provides no mechanism for analyzing whether task errors will actually be allowed to occur and whether they will impact system safety and performance. Combining CREAM with formal methods or simulation allows these issues to be addressed in what are conventionally called third generation approaches.

2.4. Third Generation HRA

Third generation HRAs implement first- and second-generation methods in simulation or formal analyses to account for dynamic system behavior in predictions. For example, SHERPA (a simulator for human error probability analysis; Di Pasquale et al. 2015), like most third generation HRAs (Di Pasquale et al., 2013), is based in first generation methods and uses simulation. By being first-generation-based, methods like SHERPA are limited by not including a cognitive model. Their simulation basis also means that it can miss critical interactions that would be considered with formal methods. To address these problems, the Systems Analysis for Formal Pharmaceutical Human Reliability (SAFPHR) (Zheng et al., 2020a,b, 2017) showed that basic (Zheng et al., 2020b, 2017) and extended (Zheng et al., 2020a) CREAM could be implemented in the PRISM probabilistic model checker (Kwiatkowska et al., 2011). When this includes a dynamic model of the system, SAFPHR was able to accurately predict the probability of undesirable outcomes in community pharmacies. These developments are powerful and made several specific recommendations that could improve community pharmacy (Zheng et al., 2020a). However, they are limited in two important ways. First, SAFPHR uses a very simple version of task analysis that is based on flow diagrams and thus does not allow for the specificity of behavior supported by EOFM. Second, SAFPHR requires analysts to manually identify the cognitive function failure for each task and thus does not dynamically determine how errors could occur.

3. Objectives

In this work, we hypothesized that the expressive power of the task-based taxonomy of human error (Bolton, 2017b) would give us enough information to associate every one of the possible errors with a cognitive function from CREAM (Table 3). This would enable us to automatically generate erroneous human behavior as part of a formal model (as was done in Bolton et al. 2019) but with an associated probability of error for each possible task deviation. In accomplishing this, we would be able to use probabilistic and/or statistical model checking to compute the probability of system safety and performance specifications being violated and for understanding the likelihood of different errors contributing to failures. In what follows, we describe how a method based on these capabilities was realized. This is followed by a demonstration / validation of our approach by applying it to a simple but realistic example, an ATM. In doing this, we were able to obtain preliminary results of the methods scalability and compare probabilistic and statistical model checking approaches. After this, we discuss the import of our findings and how they could influence future research.

4. Method

Our method for stochastically modeling erroneous human behavior in formal verifications is shown in Fig. 3. In this, an analyst has access to system information, a completed task analysis, and HRA (CPC_{Sum}s for each task). The analyst uses this to create a task model using *P*EOFM (which incorporates HRA information). He or she also creates part of a formal system model that describes the behavior of the system the human interacts with and specification properties that he or she wishes to check. Next, the analyst uses a systematic translation process that employs theory from CREAM and the task-based taxonomy of human error to incorporate the task model into the formal system model. This accounts for erroneous human behaviors and their probabilities. Then, the analysts run either probabilistic or statistical model checking (using PRISM) to verify specification properties against the formal model. This produces a verification report that, depending on the property checked and checking method, will produce a probability, CI around a probability, confirmation, or counterexample.

The three major contributions of this method relate to the extensions to EOFM required for this method, the novel translation process, and the specification properties that can be checked. We describe each of these below.

4.1. EOFM Extensions

To support the new method, changes were made to the EOFM language (Bolton et al., 2011), now Probabilistic EOFM (*P*EOFM). First, *P*EOFM only supports eight of the EOFM decompositions (those from Table 1). The sync (where decomposed actions are all performed synchronously) and com (for communicating information between humans) operators were removed because it was not clear how they would work with the probabilistic features. This issue is further explored in Section 6. Second, an optional cpcsum attribute was added to the base element (eofm) of each task (Fig. 4(a)). This was designed to set the CPC_{sum} from (3). Third, because nondeterminism in the initial value in PRISM models can limit analyses, an initial value was added to *SetValue* actions (Fig. 4(b)).

4.2. Translation

The translator, like EOFM's non-probabilistic translators (Bolton et al., 2019, 2011), interprets EOFM tasks as state machines using EOFM's normative (Fig. 1) and erroneous semantics Fig. 2 and Table 2. Older versions translated into the language of SAL (Bolton et al., 2019, 2011) and human errors were non-probabilistic and limited by an analyst-specified maximum. The new translator converts task models into the language of PRISM[2] and accounts for human error probabilities.

The key to including probabilities in translation is a mapping (Table 5) between CREAM's cognitive functions (Table 5) and the deviations from normative EOFM semantics from the task-based taxonomy of human error Fig. 2 and Table 2. Because local variable assignments are the mechanism EOFM uses for representing humans noticing and remembering things from the environment, errors associated with this represent observation errors. CREAM suggests that interpretation relates to how humans contextualize observed information. This maps to activity-level errors caused by the incorrect interpretation of strategic knowledge (activity *Precondition*s, *RepeatCondition*s, and *CompletionCondition*s; Fig. 2(a)). Planning errors in CREAM occur because people improperly form or understand a plan or task. Thus, map to any violation of the logic that people need to understand to properly execute the task: any activity-level transition error (Fig. 2(a)) that can be caused by violations of inherent or strategic knowledge conditions. According to CREAM, execution errors occur when the person is attempting to physically execute task actions. In our mapping, execution errors all occur at the action level, either through the transition of action execution states (Fig. 2(b)) or in the assignment of an output value (the first two rows of Table 2).

Translation was implemented as a Java desktop application that converts a *P*EOFM into PRISM's language as a Markov decision process using the architecture shown in Fig. 5. First, the formal EOFM representation is contained in a single module. This captures the behavior (normative and erroneous) associated with each task. This task module interacts with a module (or modules) that represent other parts of the system (Sys in Fig. 5).

[2]https://www.prismmodelchecker.org/manual/

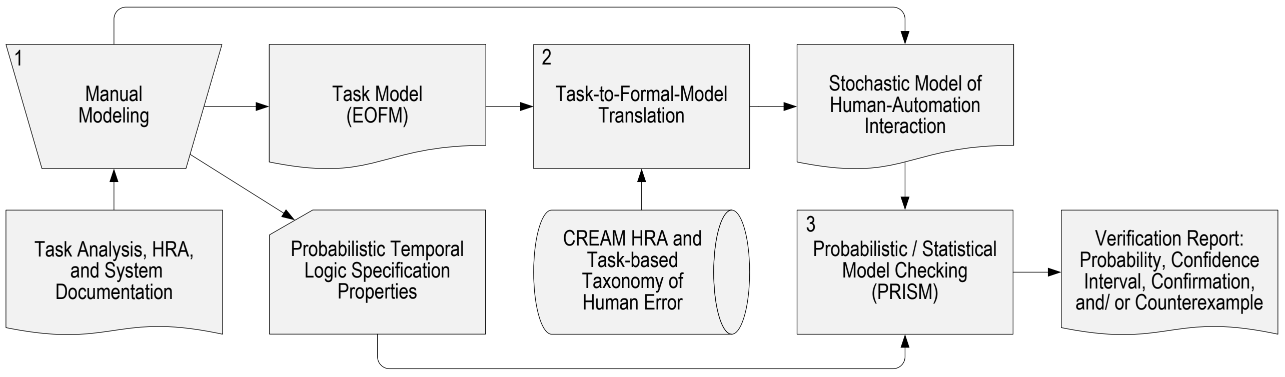

1
Manual Modeling
Task Model (EOFM)
2
Task-to-Formal-Model Translation
Stochastic Model of Human-Automation Interaction
Task Analysis, HRA, and System Documentation
Probabilistic Temporal Logic Specification Properties
CREAM HRA and Task-based Taxonomy of Human Error
3
Probabilistic / Statistical Model Checking (PRISM)
Verification Report: Probability, Confidence Interval, Confirmation, and/ or Counterexample

Figure 3: Flow diagram of our formal method for including probabilistic erroneous behavior in formal verification. Numbers in processes show order of operation.

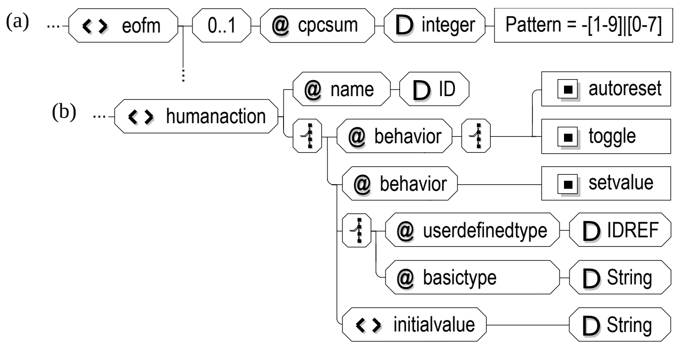


Figure 4: A visualization (SyncRO Soft SRL, 2021) of additions made to EOFM to create *P*EOFM. (a) A cpcsum attribute for each task. (b) An initial value for *SetValue* actions. A ... indicates when the remainder of the language is the same as reported in (Bolton et al., 2011).

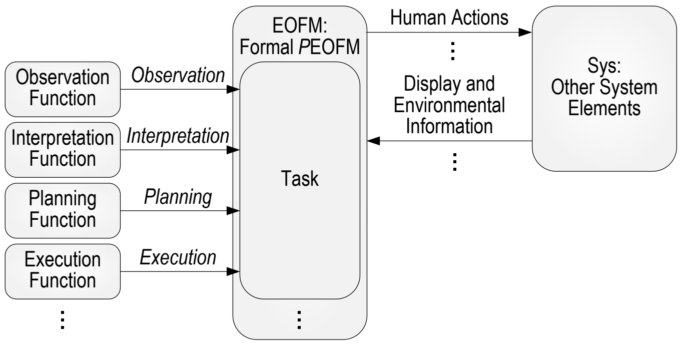


Figure 5: Architecture used for representing a *P*EOFM in PRISM. Modules are rounded rectangles. Arrows are variables shared between modules with input (destination) and output (source) relationships.

Table 5: Mapping of CREAM Cognitive Functions to Errors from EOFM's Task-based Taxonomy

Cognitive Function	Task-based Human Errors
Observation	Any action-level error related to local variable assignment, including target and value misremembrances
Interpretation	Any activity-level error (intrusion, omission, restart, or delay) caused by a violation of strategic knowledge conditions
Planning	Any activity-level error (intrusion, omission, restart, or delay)
Execution	Any action-level error that is not a local variable assignment

This can include automation behavior, the environment, or, human mission parameters as in traditional EOFM verifications (Bolton and Bass, 2010a). The interface between the human and ther parts of the system is represented by shared variables between modules. Outputs from the human task are actions and outputs from the other elements represent display information, environmental conditions, or mission perogatives. To account for the probabilities for performing tasks normatively or erroneously, the EOFM module also takes inputs from modules representing each CREAM cognitive function (Table 3). There are four such modules for each EOFM task. The probabilistic behavior of the model is implemented in these cognitive function modules. Specifically, each such module produces an output that is sent to its associated task that indicates if that function occurs with an error. Thus, at each model step, the EOFM's task model will select from the set of available behaviors based on the execution state of the task and its cognitive functions.

A task's cognitive functions compute the probability of an error (using (4) with the appropriate parameters from Table 4) as

$$P_{\text{Error}} = \begin{cases} \min\left(10^{\left(C+a\cdot CPC_{Sum}^2+b\cdot CPC_{Sum}+c\right)},1\right) & \text{if } CPC_{Sum}\geq -9 \\ 10^C & \text{otherwise.} \end{cases} \quad (5)$$

This differs slightly from (4). Because (4) does not account for situations where the equation produces probabilities greater than one, the first case of (5) selects the minimum of (4) and 1. Additionally, *P*EOFM markup allows CPC_{Sum} to go undefined (see Section 4.1). In this situation, the translator assumes a default of CPC_{Sum} that is less than -9. When this occurs, the second case of (5) assumes the cognitive function's nominal error probability (Bedford et al., 2013; Hollnagel, 1998a).

Cognitive function outputs all behave as shown in Fig. 6. All start in an *Initial* state. For any given state, the output will indicate *Error* with probability P_{Error} and *NoError* with probability $1 - P_{Error}$ in the next state. The *Initial* value allows cognitive functions to transition to *Error* or *NoError* with the appropriate probabilities before EOFM module transitions (which all require *Error* or *NoError* values) can occur.

While considering the output of the cognitive function modules, the translator interprets the semantics from Figs. 1 and 2. It does this by creating a variable in the EOFM module representing the execution state of each act and explicitly representing the transitions in Fig. 7(a) and (c) for activities and Fig. 7(b), (d), and (e) for actions. These implement the original semantics except they only allow transitions to occur if the behavior is enabled by the task's cognitive function outputs. So, for example, the transitions in Fig. 7(a) and (b) represent normative behavior (from Fig. 1) and can thus only occur when there are no errors in the associated cognitive functions. In the transitions, there is an additional new condition on *Reset* that allows it to occur when the act's parent is ready (*Parent* = *Ready*). This gives an act the option to reset after its intrusion has completed. Consistent with previous non-probabilistic translators, actions with local variable assignments automatically transition from *Ready* to *Done*, with the variable assignment occurring in the process. This is done because modules external to EOFM cannot observe this action and this form of the transition improves model scalability.

Figure 7(c)–(e) represent the erroneous behaviors from Fig. 2. Additional conditions are added to account for the mappings for observation, interpretation, planning, and execution errors from Table 5. Note that (d) and (e) describe the difference ways that errors manifest for human actions (which rely on the execution cognitive function) or local variable assignments (which depend on observation) respectively. In both of these, the *Executing*-to-*Done* transition is eliminated. This is because an action's *EndCondition* is true by default when the action is performed and action performance time is not currently modeled (actions are treated as being instantaneous). This means that an *Executing*-to-*Done* omission is functionally equivalent to a *Ready*-to-*Done* omission. Also note that in (e), *Ready*-to-*Executing* and *Done*-to-*Executing* intrusions skip the *Executing* state for the same reason that normative local variable assignments do. Finally, CREAM allows for incorrect actions to be performed (for the execution function) or the wrong object observed (for the observation one) (Table 3) as a single error for the respective function. The original erroneous transitions (Fig. 2) can support this behavior, but require both an intrusion and omissions. Thus, to represent wrong action and wrong object observed error types as a single error in our method, both (d) and (e) contain an extra transition (*Ready*-to-*Executing* in (d) and *Ready*-to-*Done* with implied execution in (e)) to allow an incorrect action or incorrect local variable assignment (respectively) to be performed when the given action should execute normatively.

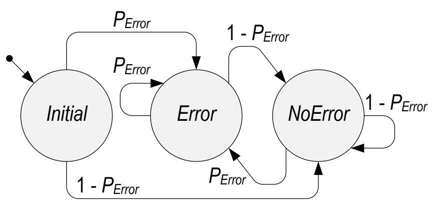


Figure 6: Transition logic for the value/state of the output variable of cognitive function modules (see Fig. 5) at each model step. These transitions are probabilistic, where P_{Error} is calculated using (5).

As in previous translators (Bolton et al., 2019, 2011), the new one creates a variable representing the output or product of each human action. For *Autoreset* and *Toggle* actions, these are Boolean. For *SetValue* actions and local variable assignments, the variables have the type specified in the *P*EOFM markup. When the normative action transitions from *Ready* to *Executing* (or from *Ready* to *Executing* with presumed execution for local variable assignments; Fig. 7(b)), the corresponding output or local variable is set to its appropriate value: true for *AutoReset*, the negation of its current value for *Toggle*, and the markup-specified value for *SetValue* and local variable assignments.

Erroneous action intrusions also exhibit this behavior, but vary in terms of what variable is assigned or what value is assigned to it. For all actions, the intrusions that transition from *Ready* with a negated start condition have the same target variable and value that would be assigned normatively. This is also true of intrusions that originate from the *Done* state. Both transitions represent a situation where an action is inserted (extraneously) into the executing task. Action intrusions originating from *Ready* with a normative (non-negated) *StartCondition* represent intrusions where the normative action is replaced with something else. Since this "something else" could be many different actions, the translator creates multiple transitions of this type. For a given human action, this means that the translator creates transitions where the human sets the output of every other possible human action instead of the one for the current action (one transition per other action). Similarly, for local variables, this means that the translator creates transitions where all of the other local variable assignments from the markup occur instead of the correct one. Additionally, these types of transitions are used for representing Action Value Substitutions and action target substitutions (for *SetValue* human actions) and value misremembrances and target misremembrances (for local variable assignments)(Table 2). For both action value substitutions and misremembrances, the translator creates multiple transitions where the value assigned to the correct variable is wrong in each possible way that it could be wrong, one transition for every possible wrong value. For action target substitutions and target misremembrances, the translator creates transitions where the incorrect variable is assigned the correct value for every possible human action output or local variable (respectively) that has the same type as the correct variable.

Analysts manually complete the sys module (Fig. 5. However, the translator creates a template for this that defines the output variables and provides a pattern for module transitions.

4.3. Specification Properties

Finally, analysts must create specifications to verify against models. Currently, these must also be manually created by the analyst. While many properties are possible, current efforts use probabilistic temporal logic specifications of the form:

$$\mathsf{P} = ?[\mathbf{F}(\mathit{FailureCondition})]. \tag{6}$$

When this specification is checked, it instructs the model checker to compute the probability ($\mathsf{P} = ?$) that eventually ($\mathbf{F}$) a failure condition (*FailureCondition*) occurs.

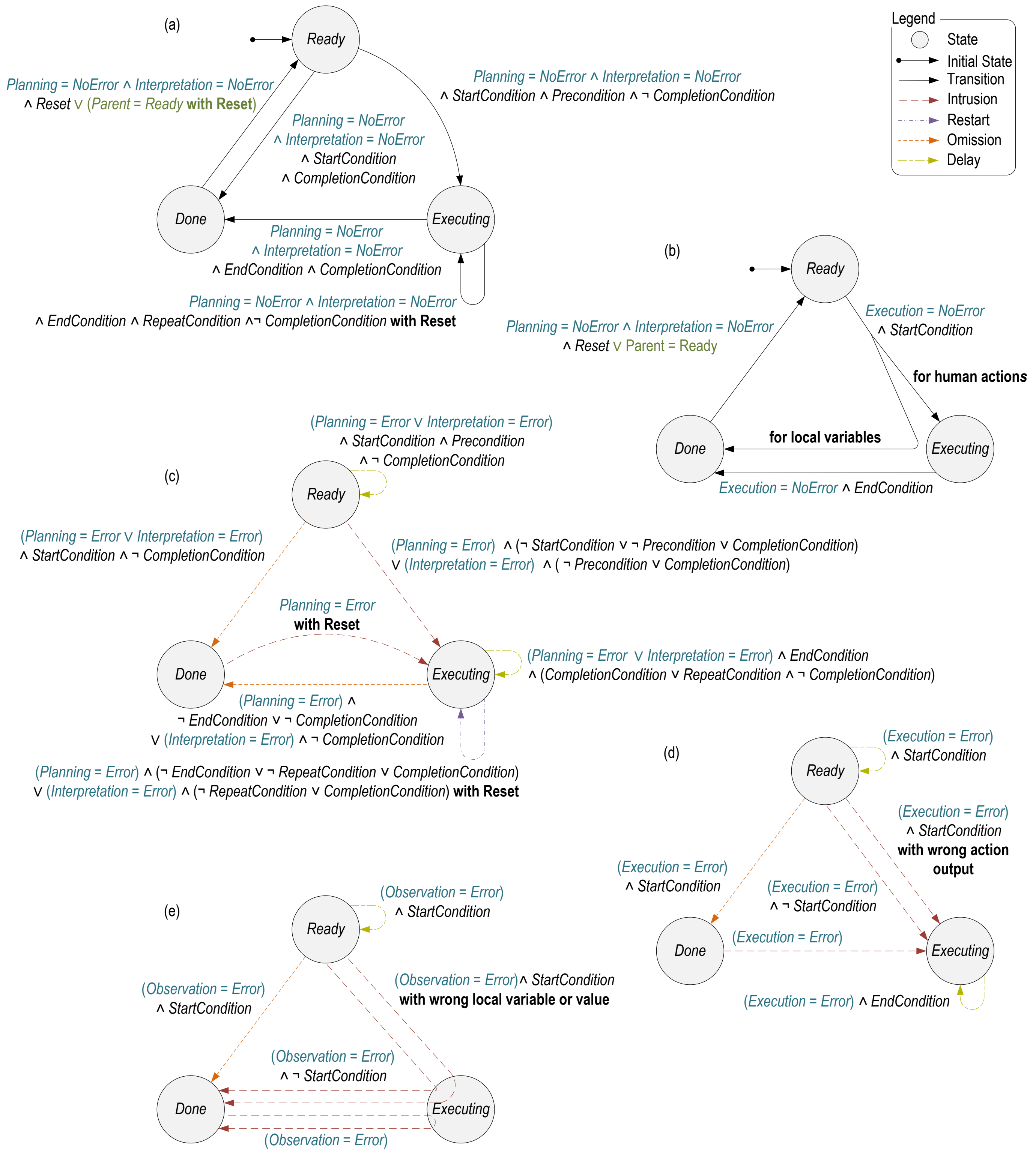

Legend
State
Initial State
Transition
Intrusion
Restart
Omission
Delay
(a)
Ready
Done
Executing
Planning = NoError ∧ Interpretation = NoError
∧ Reset ∨ (Parent = Ready with Reset)
Planning = NoError ∧ Interpretation = NoError
∧ StartCondition ∧ Precondition ∧ ¬ CompletionCondition
Planning = NoError
∧ Interpretation = NoError
∧ StartCondition
∧ CompletionCondition
Planning = NoError
∧ Interpretation = NoError
∧ EndCondition ∧ CompletionCondition
Planning = NoError ∧ Interpretation = NoError
∧ EndCondition ∧ RepeatCondition ∧¬ CompletionCondition with Reset
(b)
Ready
Done
Executing
Planning = NoError ∧ Interpretation = NoError
∧ Reset ∨ Parent = Ready
Execution = NoError
∧ StartCondition
for human actions
for local variables
Execution = NoError ∧ EndCondition
(c)
Ready
Done
Executing
(Planning = Error ∨ Interpretation = Error)
∧ StartCondition ∧ Precondition
∧ ¬ CompletionCondition
(Planning = Error ∨ Interpretation = Error)
∧ StartCondition ∧ ¬ CompletionCondition
(Planning = Error) ∧ (¬ StartCondition ∨ ¬ Precondition ∨ CompletionCondition)
∨ (Interpretation = Error) ∧ (¬ Precondition ∨ CompletionCondition)
Planning = Error
with Reset
(Planning = Error ∨ Interpretation = Error) ∧ EndCondition
∧ (CompletionCondition ∨ RepeatCondition ∧ ¬ CompletionCondition)
(Planning = Error) ∧
¬ EndCondition ∨ ¬ CompletionCondition
∨ (Interpretation = Error) ∧ ¬ CompletionCondition
(Planning = Error) ∧ (¬ EndCondition ∨ ¬ RepeatCondition ∨ CompletionCondition)
∨ (Interpretation = Error) ∧ (¬ RepeatCondition ∨ CompletionCondition) with Reset
(d)
Ready
Done
Executing
(Execution = Error)
∧ StartCondition
(Execution = Error)
∧ StartCondition
with wrong action
output
(Execution = Error)
∧ StartCondition
(Execution = Error)
∧ ¬ StartCondition
(Execution = Error)
(Execution = Error) ∧ EndCondition
(e)
Ready
Done
Executing
(Observation = Error)
∧ StartCondition
(Observation = Error)∧ StartCondition
with wrong local variable or value
(Observation = Error)
∧ StartCondition
(Observation = Error)
∧ ¬ StartCondition
(Observation = Error)

Figure 7: Transition diagram showing how the *PEOFM* to PRISM translator interprets the transition semantics from Figs. 1 and 2. Color is used in transition logic to highlight new conditions. Note that here *Observation*, *Interpretation*, *Planning*, and *Execution* represent the output variables from the associated task's cognitive function modules (Figs. 5 and 6). (a) How the translator interprets normative transitions for activities from Fig. 1(a). (b) How the translator interprets normative transitions for actions from Fig. 1(b). (c) How it interprets erroneous transitions for activities from Fig. 2(a). (d) How it interprets erroneous transitions for actions that execute human actions from Fig. 2(b). (e) How it interprets erroneous transitions for actions that execute a local variable assignment.

5. Application

As a proof of concept, we used the method to evaluate an ATM. This application was chosen because ATMs have well-documented design variations that can impact user errors. Specifically, some ATM designs can cause a post-completion error (where a human omits actions related to subsidiary goals once the primary goal is achieved (Byrne and Bovair, 1997)): a user leaving the card in the machine after receiving cash. There are also known probabilities for post-completion errors (Ratwani and Gregory Trafton, 2010; Ratwani and Trafton, 2011), enabling us to determine if our approach produces valid predictions. In what follows, we describe the behavior of two ATM variants and how they were formally modeled. We also describe the task behavior the human operator uses when interacting with the ATM. We then describe how these models were integrated into complete formal models and used to evaluate the probability errors caused problems with the system.

5.1. The Formal Model of the ATM

Figure 8 shows the formal models describing the behavior of the two variations of the ATM. In the first (Fig. 8(a)), the machine starts on a welcome screen. If the user enters a card, he or she is brought to the state for entering the pin. In this state, the user must enter his or her correct pin to progress, with incorrect pins keeping them in the current state. If the correct pin is entered, the user reaches the withdraw state. Here, the user must enter a monetary value that is greater than zero, then the machine outputs the cash, the user takes it, the machine then outputs the card. When the user takes the card, the machine returns to the welcome screen. When in the pin or withdraw states, the user can press a cancel button to immediately output the card, which can be retrieved by the user to return the machine to the welcome screen. The second machine (Fig. 8(b)) works the same as the first except that the order in which the cash and card are output is reversed. Note that the machine shown in (a) is the one that encourages post-completion error because it allows the human to take the cash (the human's primary goal) before taking the card (a subsidiary goal).

5.2. The Task Model for Interacting with the ATM

Figure 9 visualizes the *P*EOFM representing the human task behavior for withdrawing cash from both machines in Fig. 8.[3] In this, a human who wants to withdraw cash (aGetCash) must execute four sub-activities in order (as dictated by the ord decomposition). The user first inserts his or her card (with the EnterCard *AutoReset* action via the aInsertCard activity) at the welcome screen and will continue trying to do so until this screen has cleared (as dictated by the activity's strategic knowledge). Similarly, the user will enter his or her correct pin (via the EnterPin *SetValue* action) while on the EnterPin interface state under the aEnterPin activity. Next, the user enters the desired cash value while in the Withdraw interface state. Finally, the user will retrieve machine outputs (aRetrieveOutputs) by performing two activities in any possible order (an and_par decomposition) based on when the system enables them: taking the cash and taking the card. The user knows that the activity (aRetrieveOutputs) is completed (via a completion condition) if the user thinks that the cash has been retrieved.

For this analysis, we assume a $CPC_{Sum} = 4$ for the task. This was done because it represents the minimum value that would normally be associated with the strategic control mode, implying that the human has a deep understanding of the task he or she is performing and can behave strategically. Given that most people are very familiar with ATMs and the way they work, this seemed like a reasonable assumption.

5.3. Translation and Formal System Model Construction

The XML of the *P*EOFM from Fig. 9 (which was 76 lines long) was converted into the input language of the PRISM model checker, producing a representation that is 834 lines. This Sys module was then completed in this translated version in two ways: one using the behavior from Fig. 8(a) and one using the behavior from Fig. 8(b). Thus we ultimately had two formal system models for comparison: one where we would expect a higher probability for having a credit card left in it (the one based on Fig. 8(a)) and one where we would expect this to be lower (the one based on Fig. 8(b)).

5.4. Specification Properties

We evaluate our model with three specifications, all implemented using the pattern from Eq. (6). The first checked for the presence of the post-completion error outcome: the person completing the task having received cash but leaving the card:

$$\text{Card Left}: \quad P=? \left[\mathbf{F} \left(\begin{array}{l} (aGetCash = actDone) \\ \wedge \left(\begin{array}{l} iCardPresent \\ \wedge iCashOut = NoValue \end{array} \right) \end{array} \right) \right]. \tag{7}$$

This tells the model checker to calculate the probability ($P=?$) that eventually ($\mathbf{F}$) the task completes ($aGetCash = actDone$) with the cash having been retrieved (no longer being output by the machine; $iCashOut = NoValue$) and the card remaining as an output of the machine ($iCardPresent$).

The second checked for the person completing the task while leaving cash in the machine:

$$\text{Cash Left}: \quad P=? \left[\mathbf{F} \left(\begin{array}{l} (aGetCash = actDone) \\ \wedge \left(\begin{array}{l} \neg iCardPresent \\ \wedge iCashOut > NoValue \end{array} \right) \end{array} \right) \right]. \tag{8}$$

Finally, overall reliability was assessed with a property requesting the probability of either (or both) errors occurring:

$$\text{Reliability}: \quad P=? \left[\mathbf{F} \left(\begin{array}{l} (aGetCash = actDone) \\ \wedge \left(\begin{array}{l} iCardPresent \\ \vee iCashOut > NoValue \end{array} \right) \end{array} \right) \right]. \tag{9}$$

[3]Task model XML as well as all model code used for this paper can be found at `http://fhsl.eng.buffalo.edu/resources/ProbabilisticEOFM/`.

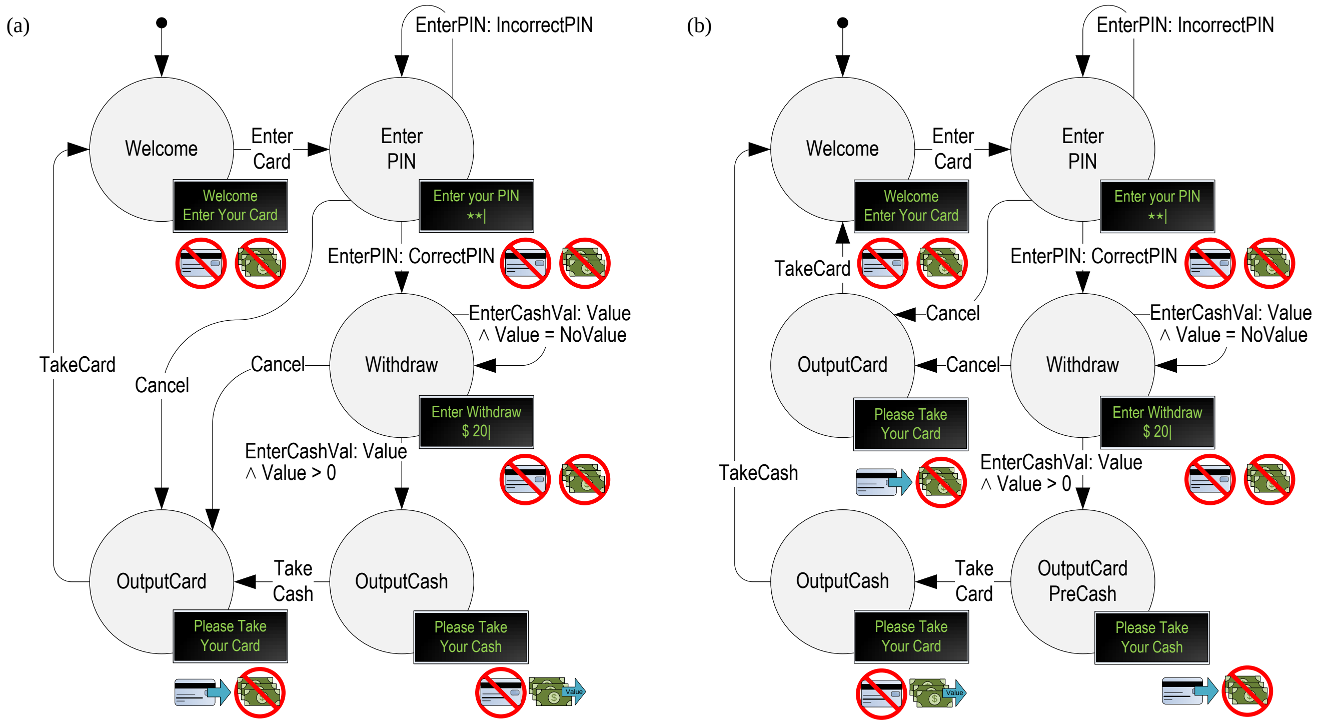
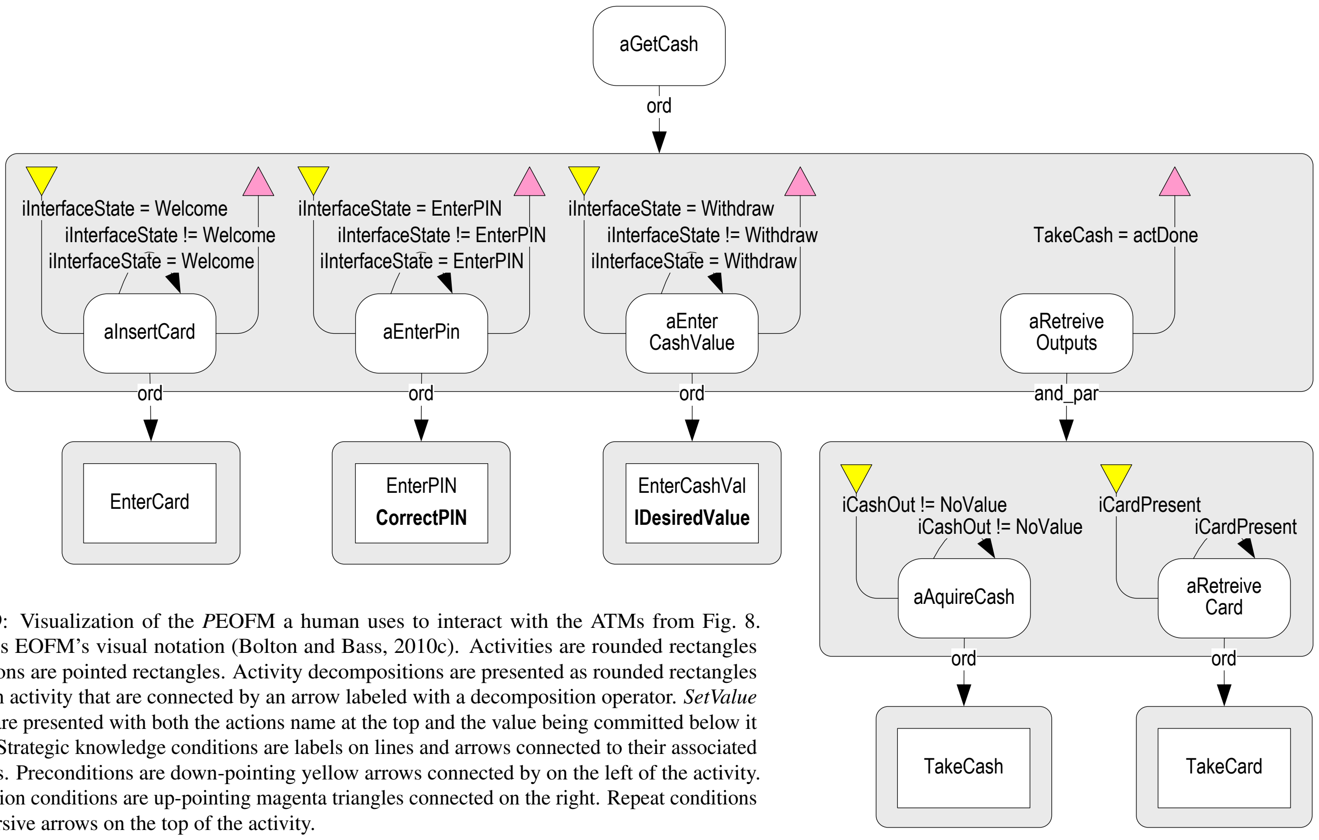


Figure 8: State machine representation of the two variations of the ATM formal model. (a) An ATM that outputs cash before outputting the user's card. (b) An ATM that outputs the user's card before outputting cash. For both figures, each state encompasses the state of the interface (iInterface; which assumes the value shown in each circle), whether or not a card is being output (iCardPresent; a Boolean variable that is false except in the OutputCard state), and the state of cash output (iCashout; which defaults to NoValue but will assume the value entered by the user in the OutputCash state).



aGetCash
ord
iInterfaceState = Welcome
iInterfaceState != Welcome
iInterfaceState = Welcome
aInsertCard
iInterfaceState = EnterPIN
iInterfaceState != EnterPIN
iInterfaceState = EnterPIN
aEnterPin
iInterfaceState = Withdraw
iInterfaceState != Withdraw
iInterfaceState = Withdraw
aEnter CashValue
TakeCash = actDone
aRetreive Outputs
ord
ord
ord
and_par
EnterCard
EnterPIN CorrectPIN
EnterCashVal IDesiredValue
iCashOut != NoValue
iCashOut != NoValue
aAquireCash
iCardPresent
iCardPresent
aRetreive Card
ord
ord
TakeCash
TakeCard

Figure 9: Visualization of the *PEOFM* a human uses to interact with the ATMs from Fig. 8. This uses EOFM's visual notation (Bolton and Bass, 2010c). Activities are rounded rectangles and actions are pointed rectangles. Activity decompositions are presented as rounded rectangles below an activity that are connected by an arrow labeled with a decomposition operator. *SetValue* actions are presented with both the actions name at the top and the value being committed below it bolded. Strategic knowledge conditions are labels on lines and arrows connected to their associated activities. Preconditions are down-pointing yellow arrows connected by on the left of the activity. Completion conditions are up-pointing magenta triangles connected on the right. Repeat conditions are recursive arrows on the top of the activity.

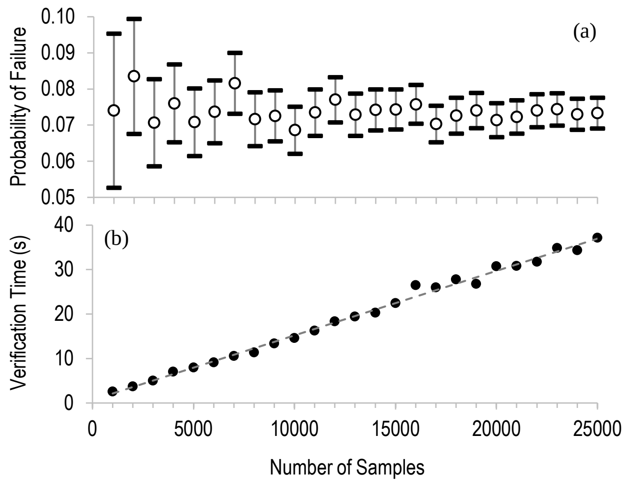


Figure 10: (a) Verification results (predicted probability and its 99% CI) and (b) times from statistical model checking for verify (9) against models containing behavior from Fig. 8(a) for different numbers of samples.

If the method accurately predicts probabilities, we would expect the model containing the ATM from Fig. 8(a) to have a higher probability for leaving the card (7) than leaving cash (8) because of the machine's encouragement of the post-completion error. We would expect the probability of leaving the card to be lower for the model using Fig. 8(b) because this should not encourage the post-completion error. We would also expect the probability of leaving the card to be similar to that of leaving the cash. Finally, we would expect the overall chance of error (9) to be higher for the model using Fig. 8(a) because of its support for the post-completion error.

5.5. *Verification and Results*

Verifications were performed on a workstation with a 12-core 3.60 GHz Intel Xeon E5-1650 with 128 Gigabytes of RAM. However, probabilistic model checking resulted in the machine running out of memory after more than 24 hours of analysis. Luckily, we were able to use statistical model checking.

Statistical checking uses samples of model traces to compute CIs on computed probabilities. This means that scalability restrictions can be avoided at the expense of accuracy. To assess this tradeoff, we conducted a small experiment using the formal system model containing the machine behavior from Fig. 8(a) and checking the overall reliability with (9). This assumed a 99% CI computed from a sample size starting at 1,000 and increasing in 1,000 increments up to 25,000. Experiment results are shown in Fig. 10. This showed the verification time increased linearly with the number of samples Fig. 10(b). It also showed that the width of the 99% CI narrowed as the sample size increased. At 25,000 samples, the verification took 37.146 seconds to estimate a probability of 0.07332 with a 99% CI of [0.069, 0.0776]. This interval is less than 0.01, or one percentage point. Thus, statistical model checking was able to produce results that we can be 99% confident are within 1 percentage point of the actual value in less than a minute of verification time. We used a sample size of 25,000 for subsequent verifications.

Figure 11 shows the results of checking each of the properties against the two model variants. This produced results that are consistent with our expectations. The model with the behavior from Fig. 11(a) showed a probability of 0.04544 of leaving the card in the machine, a value significantly higher (shown by the CIs) than it was for the machine from Fig. 11(b). The second value was also comparable to the values seen for both models for leaving cash in the machine. Finally, the machine supporting the post-completion error was significantly more likely to see any error (0.07332) than the other machine (0.06448).

6. Discussion

In this research, we introduced a new formal method that combines task modeling, a taxonomy of erroneous human behavior, and HRA. This is able to both formally generate HAI errors and account for the probabilities of these errors. This enables engineers to evaluate reactive HAI designs by using probabilistic and statistical model checking to determine the relative probability of different outcomes both between and within designs. This is a major contribution for formal analyses of human error and reliability, as previous approaches could only determine if errors were possible. As such, this paper also makes significant contributions to reliability engineering and system safety. Specifically, human behavior and human error play significant roles in failure of safety and reliability. The method introduced here gives engineers an unprecedented ability to assess the reliability of safety-critical, human-interactive systems. Thus, this work will enable designs and interventions that will significantly reduce the probability of human error causing system failures and disasters, saving lives, and protecting critical infrastructure.

The ATM application is illustrative because it shows that probability predictions were able to differentiate between two designs in ways that match with established performance: the post-completion error condition was significantly more likely with the interface that facilitated it than for the other interface and other errors. It is not entirely clear from the literature how likely an ATM post-completion error is. However, Ratwani and Gregory Trafton (2010); Ratwani and Trafton (2011) in studies

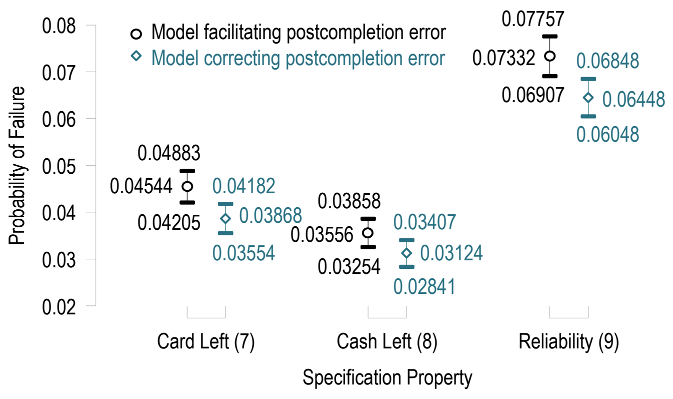


Figure 11: Verification results (prediction and 99% CI) for checking the specifications in (7)–(9) against the two variations of the model: one with the behavior for enabling a post-completion error (Fig. 8(a); black) and one with the behavior for avoiding it (Fig. 8(a); blue).

on post-completion errors found that when they are facilitated by the HAI, they occur about 5% of the time. This 5% average is consistent with the 4.544% observed with our model. Thus, while limited, the ATM application does suggest that our approach is capable of producing valid predictions.

It is important to note that the ATM example produces more than just post-completion errors. This is evidenced by the fact that the overall error rates were higher than the post-completion ones and that the model that did not allow for the post-completion error still exhibited errors. Of course, other systems will have different types of errors that will be critical to system safety and performance. Future research should seek to validate our method with other applications and human subject experiments. Additionally, there are extensions and considerations necessitated by our approach. We discuss these below.

6.1. Probability Accuracy

While the presented results are consistent with the literature, the predictions seem high. This is not necessarily surprising because the method accounts for all of the possible ways an error can manifest, including all planning errors (incorrect task knowledge). For a task as familiar as interacting with an ATM, it is conceivable that planning errors might be less likely than for other tasks. It is worth noting that the method can be easily adapted to account for different conditions. First, researchers can assess the values of the CPCs used in a given environment and use those in analyses. Additionally, in situations where an analyst is sure a given cognitive function is not a factor, he or she can easily set the default probability of error in the associated function to 0 to mitigate the function's effect.

Additionally, the method enables analysts to evaluate the impact of different conditions on performance by manipulating the CPC values. For example, CREAM has a CPC representing the quality of conditions. Because any given CPC can assume a value of -1, 0, or 1, it can have up to a 2 point impact on *CPCsum*. Any analyst wishing to see how environmental or design factors impact computed probabilities should be able to appropriately adjust CPC values and rerun analyses. This should be explored more thoroughly in subsequent research.

To some extent, the absolute accuracy of the probabilistic predictions is not critical. As long as the method is capable of determining which condition is more likely, analysts will be able to take appropriate action. Future research will investigate how well the method works for predicting probabilities both absolutely and relatively.

6.2. Deeper Support for CPC Variation

The current version of the method has a single *CPCSum* and associated cognitive functions for each task. It is conceivable that CPC values could vary for different parts of a task. It should be possible for *P*EOFM to be updated to allow for greater specificity of *CPCSum* values at the activity level and have the translator create separate cognitive functions to accommodate this. Future research should explore these developments.

6.3. Scalability

The ATM example showed that scalability is a major restriction of probabilistic model checking with *P*EOFM. The current translation approach is derived from the original method that was used for including EOFM task behavior in nonprobabilistic model checking analyses (Bolton et al., 2011). In this, the execution state of every activity and action is represented with its own variable. In traditional model checking with EOFM, significant scalability improvement can be achieved by eliminating the explicit representation of EOFM activity execution state and expressing it as a formula over the execution state of actions (Bolton et al., 2016). This improvement works because it eliminates the need for intermediate task model transitions. However, this approach is currently incompatible with the human error generation method employed here because erroneous deviations occur at the intermediate transitions of the task's activities. It is, however, theoretically possible to apply the scalability improvement method from Bolton et al. (2011) to *P*EOFM, but this would ultimately require a much more complex formulation. This is because there are an increased number of intermediary transitions necessitated by error generation and the need to account for the probabilities of these transitions. This should be explored more thoroughly in future research.

Fortunately, statistical model checking has fewer restrictions than probabilistic model checking. In fact, verification times appear to scale linearly with the number of computed samples. This provides good evidence that our method, when used with statistical checking, can scale to industrial applications. This should also be investigated further in future research.

6.4. Purely Reactive System

It is important to note that our method is most appropriate for modeling reactive systems: where system behavior occurs in direct response to human actions. This is because the cognitive-function-based model assumes that stochastic behavior only originates with the human. There are many interactive systems that are purely reactive. However, this limits the applicability of our approach. To help expand the scope of applications that could be evaluated with the method, future research should investigate how to account for stochastic behavior originating from the environment and machine automated behavior.

6.5. Non-deterministic Choice

A byproduct of using Markov decision processes and statistical model checking in our method is that nondeterministic transitions (where there are multiple allowable transitions at a given step) occur with equal probability. For example, if there are two activities in an `and_seq` decomposition who both have their precondition satisfied, there will be a 0.5 probability of each executing. While this is normally the behavior an analyst will want, there are conceivable situations where humans would be more likely to choose one behavior over another. Future research should investigate how to account for this in the method.

6.6. *The* sync *Decomposition and Human-human Interaction*

*P*EOFM and the method currently support all but two of EOFMs features. First, *P*EOFM does not allow the sync decomposition, a rare condition where the human performs multiple actions synchronously. Second, the method does not support EOFMC capabilities (the variant supporting human-human coordination and communication; Bolton 2015; Bolton and Bass 2017). While CREAM's CPCs and cognitive functions do give some insight into how to account for human-human coordination errors, they do not appear to offer a theory for modeling the probability of communication errors. Future work should investigate how to incorporate EOFMC capabilities into our method.

6.7. *Dependence of Error Prediction on Task Modeling*

Because our method uses the task-based taxonomy of human error, it is complete with respect to the phenotypes of erroneous action and slip genotypes (Bolton, 2017b). This means that it should be capable of modeling nearly any type of human error and its associated probability. However, it is important to note that the type of errors that can be generated will depend on the way that the normative task model is formulated. For example, our method should be capable of generating errors where someone puts a card into the machine with incorrect orientation, or even inserts the wrong card. However, accomplishing this would require analysts to include actions in the task model for picking up different card options and inserting cards in different orientations. Doing this ultimately requires modeler insight that may not be obvious at the time of model creation. Future work should investigate how to create guidance for modelers that enables them to include model concepts that will enable complete error prediction. Additionally, progress has been made in using formal models of affordance to identify what human actions (intended or unintended) are facilitated by the interface and environment (Abbate and Bass, 2017; Kim et al., 2010). Future research should investigate whether affordance-based action prediction can be incorporated into our method.

7. Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant Nos. 1918314 and 1918140.

References

Abbate, A.J., Bass, E.J., 2017. Modeling affordance using formal methods, in: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, SAGE Publications Sage CA: Los Angeles, CA. pp. 723–727.

Aït-Ameur, Y., Baron, M., 2006. Formal and experimental validation approaches in HCI systems design based on a shared event B model. International Journal on Software Tools for Technology Transfer 8, 547–563.

Barbosa, A., Paiva, A.C., Campos, J.C., 2011. Test case generation from mutated task models, in: Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems, ACM. pp. 175–184.

Basnyat, S., Palanque, P., Schupp, B., Wright, P., 2007. Formal socio-technical barrier modelling for safety-critical interactive systems design. Safety Science 45, 545–565.

Bastide, R., Basnyat, S., 2007. Error patterns: Systematic investigation of deviations in task models, in: Task Models and Diagrams for Users Interface Design, Springer, Berlin. pp. 109–121.

Bedford, T., Bayley, C., Revie, M., 2013. Screening, sensitivity, and uncertainty for the CREAM method of human reliability analysis. Reliability Engineering & System Safety 115, 100–110.

Bell, J., Holroyd, J., 2009. Review of human reliability assessment methods. Technical Report RR679. Health and Safety Executive. Derbyshire.

Blom, H.A.P., Stroeve, S., Daams, J., Nijhuis, H.B., 2001. Human cognition performance model based evaluation of air traffic safety, in: Proceedings of the 4th International Workshop on Human Error, Safety and System Development, Linköping. pp. 11–12.

Bolton, M.L., 2015. Model checking human–human communication protocols using task models and miscommunication generation. Journal of Aerospace Information Systems 12, 476–489.

Bolton, M.L., 2017a. Novel developments in formal methods for human factors engineering, in: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, SAGE Publications Sage CA: Los Angeles, CA. pp. 715–717.

Bolton, M.L., 2017b. A task-based taxonomy of erroneous human behavior. International Journal of Human-Computer Studies 108, 105–121.

Bolton, M.L., Bass, E.J., 2009a. Building a formal model of a human-interactive system: Insights into the integration of formal methods and human factors engineering, in: Proceedings of the 1st NASA Formal Methods Symposium, NASA Ames Research Center, Moffett Field. pp. 6–15.

Bolton, M.L., Bass, E.J., 2009b. A method for the formal verification of human interactive systems, in: Proceedings of the 53rd Annual Meeting of the Human Factors and Ergonomics Society, HFES, Santa Monica. pp. 764–768.

Bolton, M.L., Bass, E.J., 2010a. Formally verifying human-automation interaction as part of a system model: Limitations and tradeoffs. Innovations in Systems and Software Engineering: A NASA Journal 6, 219–231.

Bolton, M.L., Bass, E.J., 2010b. Using task analytic models and phenotypes of erroneous human behavior to discover system failures using model checking, in: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, SAGE Publications Sage CA: Los Angeles, CA. pp. 992–996.

Bolton, M.L., Bass, E.J., 2010c. Using task analytic models to visualize model checker counterexamples, in: Proceedings of the 2010 IEEE International Conference on Systems, Man, and Cybernetics, IEEE, Piscataway. pp. 2069–2074.

Bolton, M.L., Bass, E.J., 2011. Evaluating human-automation interaction using task analytic behavior models, strategic knowledge-based erroneous human behavior generation, and model checking, in: Proceedings of the IEEE International Conference on Systems Man and Cybernetics, IEEE, Piscataway. pp. 1788–1794.

Bolton, M.L., Bass, E.J., 2012. Using model checking to explore checklist-guided pilot behavior. International Journal of Aviation Psychology 22, 343–366.

Bolton, M.L., Bass, E.J., 2013a. Evaluating human-human communication protocols with miscommunication generation and model checking, in: Proceedings of the Fifth NASA Formal Methods Symposium. Moffett Field: NASA Ames Research Center, NASA Ames Research Center, Moffett Field. pp. 48–62.

Bolton, M.L., Bass, E.J., 2013b. Generating erroneous human behavior from strategic knowledge in task models and evaluating its impact on system safety with model checking. IEEE Transactions on Systems, Man and Cybernetics: Systems 43, 1314–1327.

Bolton, M.L., Bass, E.J., 2017. Enhanced operator function model (EOFM): A task analytic modeling formalism for including human behavior in the verification of complex systems, in: Weyers, B., Bowen, J., Dix, A., Palanque, P. (Eds.), The Handbook of Formal Methods in Human-Computer Interaction. Springer, Cham, pp. 343–377.

Bolton, M.L., Bass, E.J., Siminiceanu, R.I., 2012. Generating phenotypical erroneous human behavior to evaluate human-automation interaction using model checking. International Journal of Human-Computer Studies 70, 888–906.

Bolton, M.L., Bass, E.J., Siminiceanu, R.I., 2013. Using formal verification to evaluate human-automation interaction in safety critical systems, a review. IEEE Transactions on Systems, Man and Cybernetics: Systems 43, 488–503.

Bolton, M.L., Molinaro, K.A., Houser, A.M., 2019. A formal method for assessing the impact of task-based erroneous human behavior on system safety. Reliability Engineering & System Safety 188, 168–180.

Bolton, M.L., Siminiceanu, R.I., Bass, E.J., 2011. A systematic approach to model checking human-automation interaction using task-analytic models. IEEE Transactions on Systems, Man, and Cybernetics, Part A 41, 961–976.

Bolton, M.L., Zheng, X., Molinaro, K., Houser, A., Li, M., 2016. Improving the scalability of formal human–automation interaction verification analyses that use task-analytic models. Innovations in Systems and Software Engineering 13, 1–17.

Byrne, M.D., Bovair, S., 1997. A working memory model of a common procedural error. Cognitive Science 21, 31–61.

Castiglia, F., Giardina, M., Caravello, F.P., 2008. Fuzzy fault tree analysis in modern γ-ray industrial irradiator: Use of fuzzy version of HEART and CREAM techniques for human error evaluation, in: International Conference on Probabilistic Safety Assessment and Management.

Chen, D., Fan, Y., Li, W., Wang, Y., Zhang, S., 2019. Human reliability prediction in deep-sea sampling process of the manned submersible. Safety science 112, 1–8.

Clarke, E.M., Grumberg, O., Peled, D.A., 1999. Model Checking. MIT Press, Cambridge.

Di Pasquale, V., Iannone, R., Miranda, S., Riemma, S., 2013. An overview of human reliability analysis techniques in manufacturing operations, in: Schiraldi, M. (Ed.), Operations management. InTech, pp. 221–240.

Di Pasquale, V., Miranda, S., Iannone, R., Riemma, S., 2015. A simulator for human error probability analysis (SHERPA). Reliability Engineering & System Safety 139, 17–32.

Fields, R.E., 2001. Analysis of Erroneous Actions in the Design of Critical Systems. Ph.D. thesis. University of York. York.

Fujita, Y., Hollnagel, E., 2004. Failures without errors: Quantification of context in HRA. Reliability Engineering & System Safety 83, 145–151.

Geng, J., Murè, S., Baldissone, G., Camuncoli, G., Demichela, M., 2015. Human error probability estimation in ATEX-HMI area classification: From THERP to FUZZY CREAM. Chemical Engineering Transactions 43, 1243–1248.

Gunter, E.L., Yasmeen, A., Gunter, C.A., Nguyen, A., 2009. Specifying and analyzing workflows for automated identification and data capture, in: Proceedings of the 42nd Hawaii International Conference on System Sciences, IEEE Computer Society, Los Alatimos. pp. 1–11.

He, X., Wang, Y., Shen, Z., Huang, X., 2008. A simplified CREAM prospective quantification process and its application. Reliability Engineering & System Safety 93, 298–306.

Hollnagel, E., 1993. The phenotype of erroneous actions. International Journal of Man-Machine Studies 39, 1–32.

Hollnagel, E., 1998a. Cognitive Reliability and Error Analysis Method (CREAM). Elsevier, Oxford.

Hollnagel, E., 1998b. Context, cognition and control, in: Waern, Y. (Ed.), Co-operative Process Management, Cognition and Information Technology. Taylor & Francis, London, pp. 27–52.

Hollnagel, E., Kaarstad, M., Lee, H.C., 1999. Error mode prediction. Ergonomics 42, 1457–1471.

Kebabjian, R., 2018. Accident statistics. `http://www.planecrashinfo.com/cause.htm`.

Kenny, D.J., 2015. 26th Joseph T. Nall report: General Aviation Accidents in 2014. Technical Report. AOPA Foundation.

Kim, N., Shin, D., Wysk, R., Rothrock, L., 2010. Using finite state automata (FSA) for formal modelling of affordances in human-machine cooperative manufacturing systems. International Journal of Production Research 48, 1303–1320.

Kohn, L.T., Corrigan, J., Donaldson, M.S., 2000. To Err is Human: Building a Safer Health System. National Academy Press, Washington.

Kwiatkowska, M., Norman, G., Parker, D., 2007. Stochastic model checking, in: Bernardo, M., Hillston, J. (Eds.), Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation, Springer, Berlin. pp. 220–270.

Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems, in: International Conference on Computer Aided Verification, Springer. pp. 585–591.

Le Bot, P., 2004. Human reliability data, human error and accident models—illustration through the three mile island accident analysis. Reliability Engineering & System Safety 83, 153–167.

Manning, S.D., Rash, C.E., LeDuc, P.A., Noback, R.K., McKeon, J., 2004. The role of human causal factors in US Army unmanned aerial vehicle accidents. Technical Report 2004-11. USA Army Research Laboratory.

NHTSA, 2008. National motor vehicle crash causation survey: Report to congress. Technical Report DOT HS 811 059. National Highway Traffic Safety Administration. Springfield.

Palanque, P.A., Bastide, R., Senges, V., 1996. Validating interactive system design through the verification of formal task and system models, in: Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Chapman and Hall, London. pp. 189–212.

Pan, D., Bolton, M.L., 2018. Properties for formally assessing the performance level of human-human collaborative procedures with miscommunications and erroneous human behavior. International Journal of Industrial Ergonomics 63, 75–88.

Paternò, F., Santoro, C., 2001. Integrating model checking and HCI tools to help designers verify user interface properties, in: Proceedings of the 7th International Workshop on the Design, Specification, and Verification of Interactive Systems, Springer, Berlin. pp. 135–150.

Rantanen, E., Deeter, J., Burke, S., Wang, Y., 2012. Human factors evaluation of pharmacy operations, in: 2012 Symposium on Human Factors and Ergonomics in Health Care: Bridging the Gap, HFES, Santa Monica.

Rashed, S.K., 2016. The concept of human reliability assessment tool CREAM and its suitability for shipboard operations safety. Journal of Shipping and Ocean Engineering 6, 313–320.

Ratwani, R.M., Gregory Trafton, J., 2010. A generalized model for predicting postcompletion errors. Topics in cognitive science 2, 154–167.

Ratwani, R.M., Trafton, J.G., 2011. A real-time eye tracking system for predicting and preventing postcompletion errors. Human–Computer Interaction 26, 205–245.

Reason, J., 1990. Human Error. Cambridge University Press, New York.

Reer, B., 2008. Review of advances in human reliability analysis of errors of commission part 2: EOC quantification. Reliability Engineering & System Safety 93, 1105–1122.

Schraagen, J.M., Chipman, S.F., Shalin, V.L., 2000. Cognitive Task Analysis. Lawrence Erlbaum Associates, Inc., Philadelphia.

Stanton, N.A., Ashleigh, M.J., Roberts, A.D., Xu, F., 2001. Testing Hollnagel's contextual control model: Assessing team behaviour in a human supervisory control task. Journal of Cognitive Ergonomics 5, 21–33.

Swain, A., 1987. Accident Sequence Evaluation Program Human Reliability Analysis Procedure. Technical Report NUREG/CR-4772. US Nuclear Regulatory Commission. Washington, DC.

SyncRO Soft SRL, 2021. Relax NG schema diagram. URL: `http://www.oxygenxml.com/doc/ug-oxygen/topics/relax-ng-schema-diagram.html`.

Weyers, B., Bowen, J., Dix, A., Palanque, P. (Eds.), 2017. The Handbook of Formal Methods in Human-Computer Interaction. Springer, Berlin.

Williams, J.C., 1986. HEART – a proposed method for achieving high reliability in process operation by means of human factors engineering technology, in: Proceedings of a Symposium on the Achievement of Reliability in Operating Plant, Safety and Reliability Society (SaRS), NEC, Birmingham.

Worm, A., 2001. Breaking the barriers: Facilitating efficient command and control in multi-service emergency management, in: 8th World Conference on Emergency Management, Oslo. pp. 19–22.

Yang, Z., Bonsall, S., Wall, A., Wang, J., Usman, M., 2013. A modified CREAM to human reliability quantification in marine engineering. Ocean Engineering 58, 293–303.

Zhang, S., He, W., Chen, D., Chu, J., Fan, H., 2019. A dynmaic human reliability assessment approach for manned submersibles using PMV-CREAM. International Journal of Naval Architecture and Ocean Engineering .

Zheng, X., Bolton, M.L., Daly, C., 2020a. Extended SAFPHR (Systems Analysis for Formal Pharmaceutical Human Reliability): Two approaches based on extended cream and a comparative analysis. Safety Science 132, 18 pages. doi:`10.1016/j.ssci.2020.104944`.

Zheng, X., Bolton, M.L., Daly, C., Biltekoff, E., 2020b. The development of a next-generation human reliability analysis: Systems analysis for formal pharmaceutical human reliability (SAFPHR). Reliability Engineering & System Safety 202, 15 pages. doi:`10.1016/j.ress.2020.106927`.

Zheng, X., Bolton, M.L., Daly, C., Feng, L., 2017. A formal human reliability analysis of a community pharmacy dispensing procedure, in: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, SAGE, Los Angeles. pp. 728–732.