Automated Framework for Formal Operator Task Analysis

Ayesha Yasmeen
Department of Computer Science
University of Illinois, Urbana, Illinois, USA
yasmeen@illinois.edu

Elsa L. Gunter
Department of Computer Science
University of Illinois, Urbana, Illinois, USA
egunter@illinois.edu

ABSTRACT

Aberrant behavior of human operators in safety critical systems can lead to severe or even fatal consequences. Human operators are unique in their decision making capability, judgment and nondeterminism. There is a need for a generalized framework that can allow capturing, modeling and analyzing the interactions between computer systems and human operators where the operators are allowed to deviate from their prescribed behaviors for executing a task. This will provide a formal understanding of the robustness of a computer system against possible aberrant behaviors by its human operators. We provide a framework for (i) modeling the human operators and the computer systems; (ii) formulating tolerable human operator action variations(protection envelope); (iii) determining whether the computer system can maintain its guarantees if the human operators operate within their protection envelopes; and finally, (iv) determining robustness of the computer system under weakening of the protection envelopes. We present Tutela, a tool that assists in accomplishing the first and second step, automates the third step and modestly assists in accomplishing the fourth step.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods, Model checking, Reliability*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Model theory, Temporal logic*

General Terms

Model generation, Property satisfaction

Keywords

Task analysis, Human operator, Verification framework, System robustness, Protected task execution

1. INTRODUCTION

Computers and automated systems are omnipresent in the twenty-first century. From our workplaces to our homes, computers have become integral parts of our daily lives. However, "to err is human". Humans quite often deviate from the recommended methods of operating computer systems often in manners unexpected by the designers. From the infamous "Three Mile Island accident" [17] to patients dying from wrong doses of radiation [21], human handling and judgment has led to immense and in some cases fatal consequences. Hence there is a need for formal mechanisms for analyzing human computer interactions in safety critical systems.

Example Scenario: Let us consider a simple scenario: Everyday customers walk up to self-checkout systems at supermarkets to buy groceries. These self-checkout systems are equipped with interactive touch-screen displays, bar code scanners, bagging areas equipped with weighing scales and payment accepting devices. These equipments are controlled by a computerized system in the back end. A customer begins by pressing the word "Start". Then the system officially allows the customer to start scanning the items. After scanning, each item is placed in a bag in the bagging area where the weight of the item is matched with the expected weight. Any significant weight mismatch results in an alarm. We will ignore grocery items that do not have any bar codes on them. After all items have been scanned, the customer then presses the payment button, pays for the items and subsequently leaves the store with the items just purchased. Just as much as the customers can successfully follow these steps to buy items from the store, they can (un)willingly create system loss for the store owners. What if a customer scans one item but places another more expensive item of the same weight in the bag? What if the customer forgets to scan an item in the cart altogether? Having a formal characterization of problematic and non-problematic atypical customer behaviors will help understand, analyze and possibly improve the self-checkout system.

Formal verification provides insights into whether a computer system conforms to its desired properties. These insights are provided modulo assumptions about the environment in which the system operates. In this work, we focus on a very prominent component of the environment: the humans who operate them. Their capability of autonomous judgment, decision making and improvising actions make them a unique component of the environment. However, whether human operators should be considered as part of the environment or should be considered as another part of the system itself is an interesting issue. Consider the scenario at an airport with an air traffic control tower, where planes are landing and taking off regularly. Their arrivals and departures need to be perfectly coordinated by the pilots of the planes and the operators at the air traffic control tower to avoid accidents. The operators in the control tower can be regarded as part of the system generating safe flight path information. The pilots need to adhere to the directions being issued from the control tower. A pilot that always follows guidelines can be considered as part of the system. If the pilot makes a misjudgment or fails to follow guidelines he will cease to be a part of the system and become a part of the environment. Hence, in one single scenario, we observe two different roles for human operators. To be able to model human operator behaviors succinctly, a formal technique will need to be able to capture both natures of human behavior: the typical behavior that is part of the system and the atypical behavior that is part of the environment.

The unique features of human behaviors spawned research on *Operator Task Analysis (OTA)*, which relies on ideas from Human Factors (HF) and Human Computer Interactions (HCI) to analyze what is expected from operators in order to anticipate errors and thereby increase robustness of the system. We aim to augment this line of investigation with work on analyzing how the manner in which an operator performs a task affects the safe and effective operation of a system. We will not consider the cognitive aspects of human decision making, instead we will focus on all possible human operator actions and their effects. Current formal frameworks sometimes omits any assumptions about the human operators, sometimes they are considered as all powerful antagonists who can do absolutely anything, and at some other times they are assumed to behave perfectly and always operate within the guidelines provided to them. We intend to provide a middle ground, where the human operators are neither neglected, nor are they allowed to be unrealistically powerful antagonists. We intend to specify and analyze the combined system of computers and humans where the human operators can deviate within "tolerable" bounds from their recommended workflows.

1.1 Related Work

"Operator Task Analysis" [19] and its hierarchical version Hierarchical Task Analysis (HTA)[7] aims to determine how a task is actually accomplished by human operators, what special factors are involved in or required of the operators to accomplish the goal the task is supposed to achieve. Their goal is to analyze the possible operator action variations from a specified set of tasks depending upon environmental effects. Action Error Analysis (AEA) [30] attempts to find the possible future deviations in operator behavior. Work Safety Analysis (WSA) [34] analyzes each step in a specific task and analyzes how variations in performing that step can cause different types of hazards. All these techniques provide an informal method of identifying and analyzing human action variations. Tasks have been modeled using trees [20], real time logic [14] and predicate logic [15]. Operator Function Model (OFM) [23] provides a finite state machine based analytical tool to give a task analytic structure of operator behavior. Bolton *et al* extend OFM to EOFM [3] with task sequencing and conditional constraints to better model human task behavior. Fault Tree Analysis uses logic diagrams to analyze the failure process of a system. All these existing task analysis works mostly rely on graphical models and tools to depict atypical human task execution behavior of a system. What is lacking is a formal modeling technique for characterizing the atypical behaviors. Atypical behaviors can be both safe and unsafe depending upon the robustness of the computer system. Precise formal understanding and analysis of safe typical task execution, safe atypical task execution and unsafe atypical task execution is the goal of this work.

The usual trend in formal analysis in determining whether a system maintains a guarantee is by performing model checking [4]. Model checking a system M is determining whether M satisfies a requirement ψ denoted by $M \models \psi$. Although this works well for "closed" systems (systems with no interaction with the rest of the world), for "open" (systems that interact with their environment) systems it is not sufficient. This has given rise to the notion of "robust satisfaction" of a property [18]: a system M should satisfy a property ψ when composed with any environment E, $M||E \models \psi$. This trend is the closest to our work here. Any human operator system should be verified against all possible reasonable human behaviors. Our goal is to provide a framework to assist in formulating reasonable or protected human task execution behavior and determining that they are indeed protected behavior.

Given a definition of reasonableness of behavior we will build a model for all models conforming to that definition. Controller synthesis works are related in that regard. Controller synthesis [24] works model the interactions between a system and its environment as a discrete game. Controller synthesis aims at generating a winning strategy for the controller such that the system always wins irrespective of the behavior of the environment.

Assumption-guarantee works first introduced by Jones in [16] focus on a module or a program thread assuming a property about other modules while providing a guarantee those modules. If we consider human operators as modules then our concepts are similar to theirs. However, human behavior is complex enough to require several layers of categories of assumptions as we show in the next subsection. We assert that system designers need to be aware of all these categories of human behavior. Similarly the assumption-guarantee capturing capability of Interface Automata [6] is also related. Interface synthesis research [11, 27] is another related area. In modular program analysis, individual modules can be analyzed separately and then only interfaces are used when the overall system composed of all relevant modules are handled. Interfaces summarize acceptable call sequences for a module. An interface is safe if it only allows sequences that do not violate the internal invariants of its module. Permissive interfaces are those that allow all possible safe sequences. Full interfaces are both safe and permissive. Safe interfaces correspond to the protected human behaviors as presented in the next subsection. Full interfaces are achieved when the all safe behaviors are protected behaviors. Given a regular language description of an interface and its invariants, interface synthesis techniques provide models that accept words belonging to that language. Given the behavior guarantees expected from the human operators, we will use refinement to derive a model representing all well behaved human action sequences. Software environment generation from environment assumptions is also a related to our work [31]. The concepts of these works are very related to our work. These works focus on modular software or hardware systems. For one system module, the environment is comprised of other modules from that system. However, the primary novelty of our work is the context chosen by us: human operators that will interact with a system composed of hardware and software, concepts of categorizations of their interaction behaviors, guarantees systems can provide based on different categories of assumptions about human operators.

Another related area is that of human workflow specification with techniques like Yet Another Workflow Language (YAWL) [33] and UML [8]. Typically workflow verification entails determining whether a recommended workflow is deadlock free, live, livelock free and conforms to the desired goals. Formalisms used for modeling workflow include CSP [35, 22], Petri nets [32] etc. We used CSP to model and analyze operator workflow in our hospital AIDC project [10]. In [26] Shin *et al* use graphs and deterministic finite state automata to model and analyze human operator behavior. What we observe from all these is that workflow verification is mostly about verification of the recommended behavior of human operators. In our case the recommended behavior is one of the reasonable behaviors. The main thrust of our work is a support for a methodology for reasoning about and with protection envelopes; the recommended workflow is contained within the protection envelope. Clarke *et al* show in [5] how a human-intensive process can be specified using a process specification language Little-JIL, then translated into finite state automata and analyzed for satisfaction of desired properties. Their process descriptions can handle exceptional situations and hence is very close to our work.

1.2 Protection Envelope

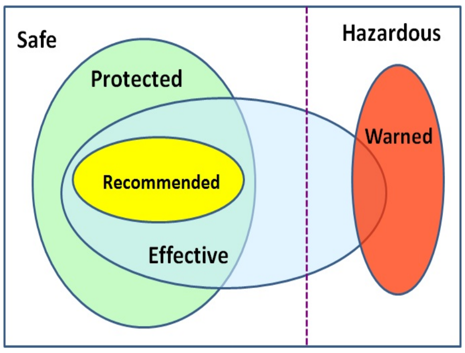


Figure 1: Protection Envelope

We now present a categorization of human behaviors. We categorize the set of all possible human behaviors among subsets shown in the Venn diagram in Figure 1. This decomposition is defined with respect to the behaviors of a complimentary set of players, in this case the computer system, the operating platform etc. For any such analysis, we need a domain-dependent concept for progress and loss. Then, the *Safe* behaviors are those in which the actions of the operator never lead to any *loss*. Behaviors that are not safe are *hazardous*. *Effective* behaviors are ones in which some progress is made. Among the effective behaviors are some desired behaviors which we refer to as *recommended* behaviors in which the operator exactly follows the steps in his task description. There may be ways to make progress that are not recommended, perhaps because the recommended procedures are just meant to describe one of many ways to get the job done or because other ways of doing the job may be hazardous. Another important set of behaviors are the *warned* behaviors, often specified in the warnings section of a user manual. These are the recognized set of hazardous behaviors. The *protected* behaviors are ones in which the operator may vary from recommended or effective behaviors without causing any hazardous consequence. The “protection envelope” is provided by an engineered set of properties of the system that form a specified subset of safe behaviors of the human operators. They are essentially the set of safe behaviors that have been identified to be safe. The “protection envelope” enforces less stringent guidelines for the human operators at the cost of a reduction in its characteristics. While the recommended behaviors are both effective and safe, the protected behaviors are guaranteed to be safe only, they may not be effective in some situations. In any scenario, the designers would like to identify all possible hazardous human behaviors thereby extending the warned behaviors to become the overall hazardous behavior set. On the other hand, designers of a system would also aim at increasing the robustness of a system against atypical human behavior by enlarging the protection envelope. An idealized situation would be where the protection envelope represents all possible safe behaviors. The notion of a protection envelope can be generalized to be extended to each and every player in a scenario. Each player has to meet some requirements on their behaviors: the human operator should behave within restrictions expected by the system developer(protection envelope); the computer system should execute to meet some requirements set by the system designer(protection envelope); the human operator may use the system along some specific step-by-step instructions (recommended task specification); the computer system will execute according to some generic specification of the system (recommended behavior).

1.2.1 Example Scenario Revisited:

Let us first define the notions of loss, effectiveness and safety in the context of the self-checkout system. The recommended human task specification is given in Figure 2. In this context (from the store owner’s point of view) a customer being able to take an item out of the store without paying for it yields loss for the store. Any behavior that does not lead to such loss is safe. For example, in the very beginning, before pressing the start button a customer can attempt to bag an item. The self-checkout system is capable of detecting this and can generate an alarm. Hence this is a safe but not recommended behavior. But if the customer leaves the store without paying for some items because they were never scanned, then the system cannot detect this. This can result in loss on part of the store and can only be avoided by the customer operating in the protected manner. A behavior is effective if a customer is able to purchase the items he placed in the cart.

1.3 The Framework

We propose a methodology to analyze computer systems and their environment components, which is comprised of four major steps: (i) identify major system components and human operators and capture knowledge about these elements including their observable actions, (ii) model the be-

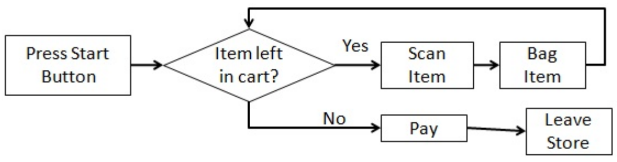


Figure 2: Customer task specification for using the Self-Scan Checkout System

havior expected from the operators, and the properties the system should maintain, (iii) verify whether the system can maintain the properties for all reasonable operator behavior i.e. any human operator behavior that conforms to their expectations, (iv) study the robustness of the system either by allowing the human operators to deviate from expected behavior or by changing their expectations. With the con-

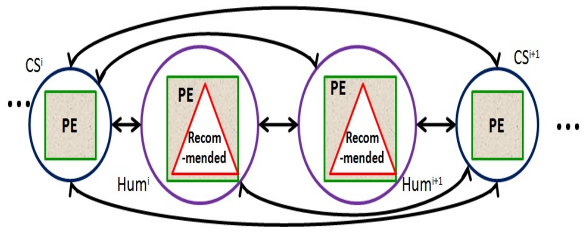


Figure 3: Computer System along with human operators

cept of different human behavior sequences given earlier, we now delineate some generic, yet prominent issues that will need to be resolved in any scenario using our framework:

1. Verifying recommended task specification

    - **Safety issue**: Is the recommended human behavior safe for the computer system? More formally, do we have Computer System + Recommended Human Behavior $\models$ Safety? We should be able to perform sanity checks or even proofs that the recommended human actions are safe.
    - **Effectiveness issue**: Does the recommended task specification ever achieve what it claims to achieve? More formally, do we have Computer System + Recommended Human Behavior $\models$ Liveness/ Effectiveness ?

2. What happens if the human somehow deviates from the recommended human operator behavior?

    - **Protection envelope**: How do we express safe variations from the recommended human behavior concisely?
    - **Verifying protection envelope**: Is the protection envelope really protected? That is, are the protected behaviors safe? More formally, do we have Computer System + Protected Human behaviors $\models$ Safety?
    - **Recommended task specification protected?** Are the recommended human behaviors protected behaviors? More formally, do we have Computer System + Recommended Human Behavior $\models$ Protection Envelope property ?
    - **Experimenting with the protection envelope**: Is there a way to methodically, easily, and efficiently experiment with and possibly enlarge the action sequence boundary specified by the protection envelope? This will allow us to expand towards the extreme boundaries of protection against anomalous human behavior.

1.3.1 Example Scenario Revisited:

Let us now fit the self checkout system described in Section 1 in our framework.

- Entity identification: The prominent agents are the customer and the self-checkout system. The observable actions of the customer are: scan an item, bag an item, pay for items etc. The behavior of the self-checkout system consists of actions like allow scanning an item, allow bagging an item etc. In order to ensure smooth execution of the system, the self-checkout system needs to store some additional information like the bar code of the items getting scanned, actual weight of items, expected weight of items, prices of items etc.
- Guarantee identification and modeling: The expected behavior (protection envelope) for the customer includes: the customer will not take out of the store items that have not been scanned. The expected behavior of the self-checkout system is that it will allow the customer to press the start button in the beginning; allow scanning of an item, bagging of an item; accept payment in the end etc. These properties will be modeled using temporal logic. The combined system of the self checkout system and the customer will be modeled using Concurrent Game Structures (Section 2).

We need to determine whether the self-checkout system will be able to maintain safety when facing any protected human behavior maintaining human guarantees. We discuss the third and fourth step in Section 2.

2. METHODOLOGY

Let us study the framework presented in Subsection 1.3. The first step suggests that the formal methodology should be able to capture different identities and their action vocabularies. In the second step, we see that the formal methodology needs to be capable of expressing the expected guarantees of behavior for each entity identified in the first step. The third step poses these problems: how do we verify that the system can maintain its guarantees in the face of all possible reasonable behavior from its human operators? What is the model for the humans and the system? How do we model and analyze all possible human behaviors? The fourth step requires the methodology chosen in the third step to be robust enough so that we can easily modify the definition of “reasonableness” or the protection envelope property and perform the same analysis without changing anything else. In summary we need a formalism that will (i) allow distinguishing among different entities (first step of framework), (ii) allow specifying the vocabulary of those entities (first step of framework), (iii) allow specifying properties on top of the formalism for each entity (first step of framework), (iv) allow modeling entity behavior (third step of framework), (v) allow modeling the interactions among the entities (third step of framework) (vi) allow reasoning about all possible reasonable behavior (third step of framework), (vii) allow efficient and simple way of re-performing the entire analysis with variations of the protection envelope (fourth step of framework). One formalism that meets our criteria is Concurrent Game Structures (CGS) [1].

Concurrent Game Structures: A CGS is an 8-tuple, $\langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ where the components are as follows: P is a finite set of players. Q is a finite set of states and $q_0 \in Q$ is the initial state. We have abstract actions Σ instead of the numbers that are used as place holders for actions in [1]. Σ is a finite nonempty set of actions for the players where the silent action $\tau \in \Sigma$. Π is a finite set of propositional atoms that help identify the states. $\pi : Q \rightarrow 2^{\Pi}$ is a function that gives the set of propositional atoms that hold of a given state. The function $e : P \times Q \rightarrow 2^{\Sigma}$ gives which actions are enabled for each player in each state. We denote the action choices available to a player p in a state q by $e_p(q)$. We enforce that the silent action τ is enabled for each player in each state: $\forall q \in Q. \forall p \in P. \tau \in e_p(q)$. We will use Σ_p to denote the vocabulary of player p where $\Sigma_p = \bigcup_{q \in Q} e_p(q)$. We will denote the action choice for a player p in a action choice vector $\bar{v}$ as $\bar{v}|_p$. The function $\delta : Q \times (e_{p_1}(q) \times e_{p_2}(q) \times \ldots e_{p_{|P|}}(q)) \rightarrow Q$ that defines state transitions. That is, given a state q, and a vector of actions $\bar{v} \in \Sigma^P$ where $\bar{v}|_p \in e_p(q)$ for each p, the value $\delta(q, \bar{v})$ is the next state q'. We require that $\forall q \in Q. \delta(q, \tau \times \ldots \times \tau) = q$. The reason behind enabling the τ action for each player in each state and $\bar{\tau}$ self-loops in each state is based on the perception that human operators cannot be assumed to always be synchronized with the computer systems. They may idle away for some time before performing an action. Given a finite or infinite sequence of states λ, we will denote the i-th state in the sequence as λ_i. We will use $\lambda_{i_1}^{i_2}$ to denote the sequence of states starting from position i_1 to position i_2. A *q-computation* r of a CGS C is an infinite sequence of states where $r_0 = q$ and for each $i \geq 0$, there is a vector $a_i \in e_{p_1}(r_i) \times e_{p_2}(r_i) \times \ldots \times e_{p_{|P|}}(r_i)$ of actions such that $r_{i+1} = \delta(r_i, a_i)$. Let us now fit CGS in our framework:

2.1 Identification and Information Collection Step

CGS will allow us to easily model all the identifiable entities, such as the computer system and the human operators as players operating concurrently. The actions available to a player across the states will capture their vocabulary. Referring to the self-scan checkout system scenario, we can have `Self-scan system` and `Customer` as the players. Each action of the self-scan system and the customer can be translated into abstract actions. For example the actions of the customer like scanning an item, bagging an item, paying for items etc can cause the following actions to be in the vocabulary of the customer: `scanitem, bagitem, payforitems`.

2.2 Protection Envelope and Safety Property Specification Step

The protection envelope is a collection of protected or reasonable behaviors. It is much easier for the system designer to provide the protection envelope using a property describing the protection criteria. Every behavior conforming to the criteria will be a protected behavior. Now we need to determine what logic the property should be expressed in. The protection envelope properties are, in fact, a form of safety property. The protection envelope property for each human operator needs to state that in every state along a protected action sequence, the human operator will guarantee nonviolation of a criterion. The criterion my very well be of the form that the human operator will not perform certain task until some other enabling task has been performed. So protection envelope properties are temporal in nature, and can be expressed using Linear Temporal Logic [25]. In fact protection envelope properties are a form of safety properties [28]. Safety properties state that something bad does not happen and if there exists a counter-example to a safety property it must be a finite one. In our work we will consider special form of safety properties called "invariants" [28] which are of the form $\Box\varphi$, where $\varphi = p$ (atomic proposition) $\mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \, \mathcal{U} \, \varphi_2 \mid \Diamond\varphi \mid \Box\varphi$. Invariants need to hold in the initial state and along each trace emanating from the initial trace. The protection envelope properties will also be of the form $\Box\varphi$. The protection envelope for the customer checking out items in a supermarket using LTL: $\Box((\neg\texttt{baggedItem}(i) \, \mathcal{U} \, \texttt{scannedItem}(i)) \wedge (\neg\texttt{itemOutofStore}(i) \, \mathcal{U} \, \texttt{itemPaidFor}(i))))$. Here `baggedIte -m`(i), `scannedItem`(i), `itemOutofStore`(i) and `itemPaid-For`(i) are atomic propositions.

2.3 Assessing Robustness of the Computer System w.r.t the Protected Behavior

In a CGS, in each state, each player chooses an action to execute and their combined actions allow the model to transition from one state to another. Thus the execution trace or computations of CGSs can encode the interactions of different entities. We first show how we can model the behavior of entities using CGSs.

2.3.1 Human Behavior Model

The e function provides the enabled action choices for each player in each state in a CGS. Here we view a human behavior as the set of actions that the human may perform in a state. Hence, we observe that the human operator behaviors can be viewed as refinements of the e function. The scenario in Figure 1 is completed when the human operator is placed in conjunction with the computer system. The combined interaction between each human behavior and the computer system can be modeled using CGSs, where the behavior of the human operator will dictate the set of possible actions for the operator in each state: the e function. Now that we have a method of modeling human behavior, we need to determine how best we can analyze all possible protected human behaviors.

2.3.2 Maximal Model

We first need to be able to model all possible protected behaviors. Then we need to assess their safety preservation capability. However, formulating and analyzing the protected behaviors one by one may become cumbersome. Given only the protection envelope property, we will have to generate each protected behavior and then analyze it. Hence we ask: can we get around the task of verifying a model for each and every possible environment and find a representative model such that satisfaction of that model in conjunction with that representative protected behavior guarantees satisfaction of that property for any behavior? Grumberg *et al* suggest searching for an ordering among environments such that given an ordering relation $\sqsubseteq$, a model M, a property ϕ, if we can find a maximal model of the environment E such that if $E \parallel M \models \phi$ then for each environment $E' \sqsubseteq E$, we have that $E' \parallel M \models \phi$ [9]. The motivation behind this approach is that now given a preordering relation, a maximal model for that relation, determining satisfaction of an environment is determined by simply determining preorder preservation. Our approach very closely follows the idea presented by Grumberg *et al.* We first define an ordering relation among operator behaviors. Then we show how we can find a maximal protected behavior such that all other protected behaviors are related to it by the ordering relation. Then we show that this maximal behavior can actually represent all protected behaviors: if the maximal behavior allows the system to operate safely then all protected behaviors will also allow the system to operate safely.

2.3.3 Ordering Relation

We observe that in a CGS, the behavior of each player is completely defined by the action choices available to it in different states. When we want to analyze the human behaviors, we are interested in the actions they take while operating a system. Hence, any ordering relation that can model human behavior will involve some restrictions on the actions available to the human operator in different scenarios. A maximal model of a human will have the maximal set of actions available to it. Any human behavior which is a sub-behavior of that human behavior will have fewer action options. And if the maximal human behavior with all its potential for causing trouble in the maximal number of ways available to it via the available actions, can still satisfy the desired property, then any sub-maximal behavior will certainly satisfy the property as it will have fewer options to cause trouble to the system. The characterization of human operator behavior using e function suggest that the more constrained and restricted the behavior, the fewer action options an operator has in each state. This leads to a notion of one behavior being contained in another behavior. Behavior containment can be modeled with one CGS having a more restricted set of action options for a set of players than another CGS. We define a CGS being a subaction CGS of another CGS as follows:

Definition Let C be a CGS where $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ and $\mathcal{P} \subseteq P$ be a set of players. $C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e', \delta\rangle$ is a $\mathcal{P}$-subaction CGS of C denoted by $C' \sqsubseteq_{\mathcal{P}} C$ if we have that $\forall q \in Q. \forall p \in P.$ if $p \in \mathcal{P}$ then $e'_p(q) \subseteq e_p(q)$ else $e'_p(q) = e_p(q)$.

Notice that $\sqsubseteq_{\mathcal{P}}$ is a transitive relation. The notion of behavior ordering through behavior containment raises the question of whether there exists a maximal protected behavior satisfying a specific restriction such that all of its subbehaviors will also satisfy the restriction. In other words, given a protection envelope property, can we have a maximal CGS such that if it satisfies a safety specification then each sub-action CGS of it will also satisfy the specification?

We are now in a quest for a maximal representative behavior respecting a property. The concept of subaction CGS suggests that if we consider the CGS where the human operator can behave without any constraint and restrict it using the property under consideration, then we will be able to arrive at a behavior which has the least restricted behavior while preserving the property. We will refer to the CGS comprising all possible human operator behavior and corresponding system responses as C_{all}.

2.3.4 C_{all}

We focus on the resilience of a computer system while facing atypical human behavior. C_{all} is a CGS which contains the interactions among system components and human operators where at each state, the human operators can execute any action that is physically possible by them. In every state in a C_{all}, each physically possible human action is enabled in each state. Hence the enabled actions and the transitions must contain the responses of the computer system for each possible human action. Usually at each state of a system there is a small subset of human actions that is expected by the computer system. The rest of the human actions are either physically impossible or will lead to an error state. We will present our tool in Section 2.5 which is capable of generating the C_{all} from a small subset of interactions among computer systems and human operators. We require that for each state the description includes (i) transitions involving safe enabled human actions and corresponding computer system reactions, (ii) physically impossible human actions. Let us consider the self-scan checkout system again. In Figure 4 we present a sparse C_{all} model provided for the initial state. In the figure, the label of each state appears next to it. Let a_1 be a computer system action and a_2 be a human action. The pair of actions (a_1, a_2) is mentioned next to the transitions they cause. In the initial state, the customers cannot pay for an item as the self-scan system displays usually do not provide any pay button at this point. This is indicated by a transition to a *dummy* state: `DisabledAction`. This dummy state is only used to help denote the impossibility of the action, it is not part of the state space. The customer may idle away the time or they may press the start button. However, the customers may also attempt to scan an item, bag an item or leave the store without scanning and paying for any item. All of these actions will result in an erroneous situation. For the sake of simplicity we use only one error state for all types of erroneous conditions. In order to reduce the burden of providing the entire C_{all}, our tool does not require specification of any transition that leads to error states. The transitions emanating from the initial state in C_{all} is presented in Figure 5. We combine the action vectors leading to the error state in Figure 5 due to space restrictions.

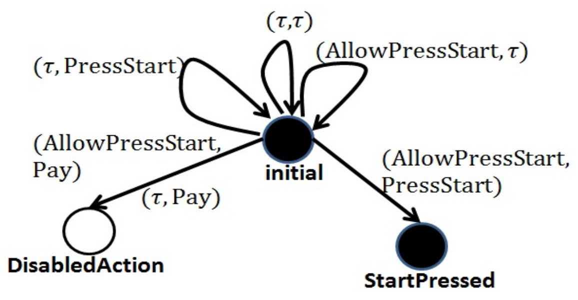
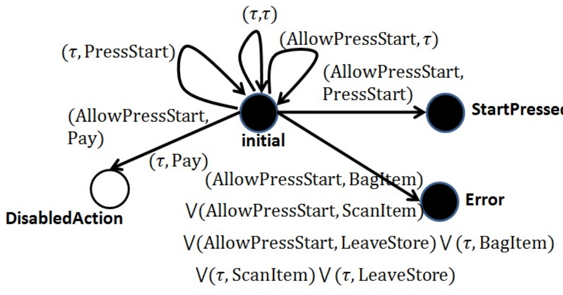


Figure 4: Initial state in incomplete C_{all}

A C_{all} is usually not the maximal protected behavior we need to find. Since our ordering relation is based on player action set ordering, we will need to trim player actions in C_{all} to obtain the maximal protected behavior. Intuitively,



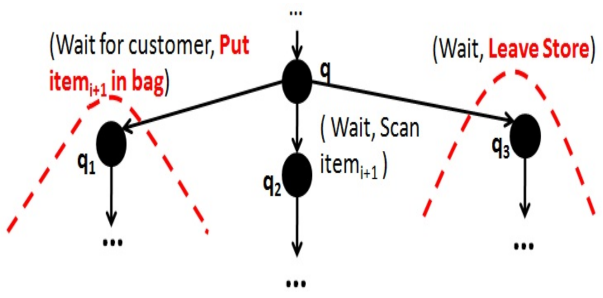
(τ,τ)
$(\tau,\text{PressStart})$
$(\text{AllowPressStart},\tau)$
$(\text{AllowPressStart}, \text{PressStart})$
StartPressed
$(\text{AllowPressStart}, \text{Pay})$
initial
(τ,Pay)
$(\text{AllowPressStart}, \text{BagItem})$
Error
$\vee(\text{AllowPressStart}, \text{ScanItem})$
DisabledAction
$\vee(\text{AllowPressStart}, \text{LeaveStore}) \vee (\tau, \text{BagItem})$
$\vee(\tau, \text{ScanItem}) \vee (\tau, \text{LeaveStore})$

Figure 5: Initial state in a complete C_{all}

the actions of the player whose behavior is getting analyzed should be modified. Since protection envelope properties are safety properties, we should trim those actions of the player which makes him do something *bad*. The easiest way of determining "badness" is if the action choice helps the CGS land in a state where a protection property is violated. Given a CGS C, a behavior restriction φ for a player p, we first determine the set of bad states $\mathcal{B}$: the states that are unsafe or potentially unsafe. $\mathcal{B}(C,p,\varphi) = \{q \mid \eta(C,p,q,\varphi)\}$. Here $\eta(C,p,q,\varphi) = (q \not\models_C \varphi) \vee (\exists \bar{v}.\, \bar{v}|_p = \tau \wedge \eta(C,p,\delta_C(q,\bar{v}),\varphi))$. The bad states are those states where the specification is either immediately violated or where the operator has performed such an action that he needs to rely on the other players helping him out to preserve the specification. Let us consider Figure 6. Here the customer has pressed the start button. Let us assume he has a non-empty cart. Then according to the recommended task specification he should scan an item from the cart. If he puts an item in the bags instead, then it violates the protection property as specified in Section 2.2. This item has been placed in the bags before having been scanned. If the customer leaves the store with the items in his cart then items are leaving the store without being paid for. Thus both q_1 and q_3 are bad states.

2.3.5 Thinning

Asarin *et al* thin actions available to a player to find a winning strategy in a game automata [2]. In a similar fashion, we "thin" the actions available to player p in an unrestricted behavior C_{all} to obtain a maximal protected behavior.



...
(Wait for customer, Put item$_{i+1}$ in bag)
q
(Wait, Leave Store)
(Wait, Scan item$_{i+1}$)
q$_1$
q$_2$
q$_3$
...
...
...

Figure 6: Thinning C_{all}

Definition Given a CGS $C_{\text{all}} = \langle P, A, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$, a player $p \in P$, a protection envelope formula $\varphi = \Box\varphi_p$, the thinning of C_{all} according to $\Box\varphi_p$ denoted by $\Theta^{\varphi}_{C_{\text{all}}}$ provides a CGS $C = \langle P, A, Q, q_0, \Sigma, \Pi, \pi, e', \delta'\rangle$, where e' is defined as follows: $\forall q \in Q.\, \forall \bar{v} \in \prod_{p' \in P} e_{p'}(q)$. if $\delta(q,\bar{v}) = q'$ then (if $q' \in \mathcal{B}(C,p,\varphi)$ then $e'_p(q) = e_p(q) \setminus \{\bar{v}_p\}$) else $e'_p(q) = e_p(q)$ and $\forall q \in Q.\, \forall p' \in P \setminus \{p\}.\, e'_{p'}(q) = e_{p'}(q)$. Thus the transitions from state q where the action vector $\bar{v}'$ has $\bar{v}'|_p = \bar{v}|_p$ gets removed in δ'.

Basically the definition states that from each state in the given CGS, if a transition leads to a bad state, then remove the action for the player p involved in that transition to obtain the refined CGS.

2.3.6 Example Scenario Revisited

Let us consider the supermarket checkout scenario. We focus on the state q after an item has been scanned in Figure 6. Here, the customer will have many action options in C_{all}. He can put the scanned item in a bag, scan another item, attempt to pay for the scanned item or even worse, leave the store along with the item before paying for it. The protection envelope states that the customer will not take an item out of the store before paying for it. So a protected behavior should not have the `LeaveStore` action enabled for the customer in state q. The mechanical infrastructure of the supermarket is incapable of preventing this action. Only the customer himself has the capability of guaranteeing that such loss for the store will not occur. Since the protection envelope of the customer is under consideration at this point and only the customer himself can guarantee safe operation, only his set of enabled actions should be reduced. This also illustrates our reasoning behind choosing CGSs over Finite State Automata. In CGSs we can very easily examine the set of actions available to players in each state and manipulate the set as necessary.

2.3.7 Thinned Model is the Maximal Model

We can now relate the thinning of a CGS according to a specification φ with obtaining a maximal model from that CGS according to φ. A thinned CGS is useful to us only if it can act as a representative for all possible protected human behavior. We need to thin only those states that are reachable from the initial state. We will achieve this by thinning the operator actions in only those states that appear in the q_0-computations in a CGS. Let us define a reachability predicate $\Upsilon_C(q_1,q_2)$ as $(q_1 = q_2) \vee (\exists q' \in Q.\, \exists \bar{v} \in \prod_{p \in P} e_p(q_2).\, \delta(q_2,\bar{v}) = q' \wedge \Upsilon_C(q_1,q'))$. Note that Υ_C is a transitive function.

Definition The fragment of a CGS $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ that is reachable from the initial state $q \in Q$ is another CGS denoted by $\mathcal{R}(C,q) = \langle P, Q, q_0, \Sigma, \Pi, \pi, e', \delta\rangle$ where $\forall q' \in Q. \forall p \in P.$ if $\Upsilon_C(q',q)$ then $e'_p(q') = e_p(q')$ otherwise $e'_p(q') = \{\tau\}$.

Now that we have defined the initial state reachable fragment of a CGS, let us define property invariance. A property ψ is an invariant for a CGS C with state space Q denoted by $\text{Inv}(C,\psi)$, if we have $\forall q \in Q. \Upsilon_C(q, q_{\text{init}}) \rightarrow q \models_C \psi$. The following lemma states that the protection envelope property holds at every state reachable from the initial state in the thinned CGS. Thus every q_0-computation satisfies the protection property in a thinned CGS.

LEMMA 2.1. *Let $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ be a CGS. Let $p \in P$ be a player, φ be the protection envelope for p.*

Let Θ^{φ}_{C} be the CGS obtained by thinning C according to φ. Then we have $\mathrm{Inv}(\mathcal{R}(\Theta^{\varphi}_{C}, q_0), \psi)$.

This lemma can be proved from the definition of thinning, reachability and property invariance and the fact that the protection envelope properties are always of the form $\Box\phi$ where ϕ is an LTL formula. We now present a lemma which correlates the initial state reachable and protection property invariant fragment of any protected behavior with the thinned behavior.

LEMMA 2.2. *Let $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ be a CGS. Let $p \in P$ be a player, φ be the protection envelope for p. Let Θ^{φ}_{C} be the CGS obtained by thinning C according to φ. Then each CGS C' such that C' is a p-subaction CGS of C and φ holds at every state reachable from q_0 in C', the portion of C' reachable from the initial state is a p-subaction CGS of Θ^{φ}_{C}. Mathematically speaking: $\forall C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$. $\forall p \in P$. $\forall \varphi$. $\forall q \in Q$. $\exists \Theta^{\varphi}_{C} = \langle P, Q, q_0, \Sigma, \Pi, \pi, e^{\Theta}, \delta\rangle$. $\forall C'$. $(\mathcal{R}(C', q_0) \sqsubseteq_{\{p\}} C \wedge \mathrm{Inv}(\mathcal{R}(C', q_0), \varphi) \Rightarrow \mathcal{R}(C', q_0) \sqsubseteq_{\{p\}} \Theta^{\varphi}_{C})$.*

PROOF. We will prove Lemma 2.2 by contradiction. Let us assume that there exists a CGS $C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e', \delta\rangle$ such that for a player $p \in P$, a property φ, $\mathcal{R}(C', q_0)$ is p-subaction CGS of C and φ is true in every state reachable from q_0 in C' but the initial state reachable portion of C' is not a p-subaction CGS of Θ^{φ}_{C}.

Since $\mathcal{R}(C', q_0) \sqsubseteq_{\{p\}} C$ and $\mathcal{R}(C', q_0) \not\sqsubseteq_{\{p\}} \Theta^{\varphi}_{C}$, we have $\exists q \in Q$. $\exists a \in \Sigma_p . \Upsilon_{C'}(q, q_0) \wedge a \in e'_p(q) \wedge a \notin e^{\Theta}_{p}(q)$. Since $\mathcal{R}(C', q_0) \sqsubseteq_{\{p\}} C$, we must have $a \in e_p(q)$. Now as $\Theta^{\varphi}_{C} \sqsubseteq_{\{p\}} C$, we must have that $a \notin e^{\Theta}_{p}(q)$ due to $\exists \bar{v} \in \prod_{p' \in P} e_{p'}(q) . \delta(q, \bar{v}) = q' \wedge q' \in \mathcal{B}(C, p, \varphi) \wedge a = \bar{v}|_p$. On one hand, if $q' \in \mathcal{B}(C, p, \varphi)$ because $q' \not\models_C \varphi$ then we have $\Upsilon_{\mathcal{R}(C', q_0)}(q, q_0) \wedge \Upsilon_{\mathcal{R}(C', q_0)}(q', q)$. This provides a state q' such that $\Upsilon_{\mathcal{R}(C', q_0)}(q', q_0)$ but $q' \not\models_C \varphi$ which contradicts our assumption of $\mathrm{Inv}(C', \varphi)$. On the other hand, let $q' \in \mathcal{B}(C_{\mathrm{all}}, p, \varphi)$ because $\exists \bar{v} . \bar{v}|_p = a \wedge \delta(q, \bar{v}) = q' \wedge \exists q_1, q_2, \ldots, q_k \in Q . q_1 = q' . \exists \bar{v_1}, \bar{v_2}, \ldots, \bar{v_k} . \forall 1 \leq i \leq k . \bar{v_i}|_p = \tau \wedge \delta(q_1, \bar{v_1}) = q_2, \ldots, \delta(q_{k-1}, \bar{v_{k-1}}) = q_k \wedge q_k \not\models_C \varphi$. Using $\Upsilon_{\mathcal{R}(C', q_0)}(q, q_0)$, $\Upsilon_{\mathcal{R}(C', q_0)}(q', q)$ and the $\bar{v_i}$s as witnesses, we obtain $\Upsilon_{\mathcal{R}(C', q_0)}(q_k, q_0) \wedge q_k \not\models_C \varphi$. This again contradicts our assumption that $\mathrm{Inv}(C', \varphi)$. $\Box$

Now we show that any sub-behavior of the maximal behavior obtained by thinning will satisfy a safety property if the maximal behavior satisfies it. This states that satisfaction of a safety property is closed under behavior ordering. This theorem enables us to state that we only need to determine whether the maximal model satisfies a safety property. All other protected behaviors will be sub-behaviors of the maximal model obtained by thinning and will satisfy the same safety property.

THEOREM 2.3. *Let $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ be a CGS, $p \in P$ be a player, φ be the protection envelope property for p. Let Θ^{φ}_{C} be the thinned maximal protected CGS. Then each CGS C' that is a p-subaction CGS of Θ^{φ}_{C} will satisfy a safety property $\psi = \Box\psi'$ if Θ^{φ}_{C} satisfies it. Mathematically: $\forall C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$. $(C' \sqsubseteq_{\{p\}} \Theta^{\varphi}_{C}) \rightarrow (\forall \psi = \Box\psi' . \ q_0 \models_{\Theta^{\varphi}_{C}} \psi \rightarrow q_0 \models_{C'} \psi)$.*

PROOF. (Sketch) Let us consider a safety property $\psi' = \Box\psi$. Let the thinned CGS Θ^{φ}_{C} obtained by thinning C according to φ satisfies ψ' *i.e.* $q_0 \models_{\Theta^{\varphi}_{C}} \psi'$. This implies that each q_0-computation will satisfy $\Box\psi$ in Θ^{φ}_{C}. Let us consider a CGS C' such that $C' \sqsubseteq_{\{p\}} \Theta^{\varphi}_{C}$. Since p has possibly fewer actions enabled at each state in C' possibly fewer transitions are possible in each state in C'. Hence each q_0-computation in C' is also a q_0-computation in Θ^{φ}_{C}. Hence they will satisfy $\Box\psi$ as well. $\Box$

We now show how to resolve the issues raised in Section 1.3. Let us consider a human operator p whose recommended task specification has been translated into a CGS C_{rec}. Let us assume a C_{all} with initial state q_0 for the combination of the human operator and a computer system is given. Let the protection envelope property for p is `PE` and the safety property for the system is `Safety`.

- **Are protected behaviors safe?** To determine whether the computer system can operate safely while facing any protected behavior expressed by p while executing a task we need to determine whether $q_0 \models_{\Theta^{\mathtt{PE}}_{C_{\mathrm{all}}}}$ `Safety` where `Safety` is of the form $\Box\psi$ (ψ as defined in Subsection 2.2).

- **Is recommended behavior protected and safe?** Then the recommended task specification is protected and safe if we have $C_{\mathrm{rec}} \models$ `PE` and $C_{\mathrm{rec}} \models$ `Safety`. However given our framework all we need to determine is whether $C_{\mathrm{rec}} \sqsubseteq_{\{p\}} \Theta^{\mathtt{PE}}_{C_{\mathrm{all}}}$. If C_{rec} is a p-subaction CGS of $\Theta^{\mathtt{PE}}_{C_{\mathrm{all}}}$ then by Theorem 2.3 we have that $C_{\mathrm{rec}} \models$ `PE` as `PE` is a safety property and if the thinned maximal protected CGS $\Theta^{\mathtt{PE}}_{C_{\mathrm{all}}}$ is safe then so is the recommended behavior CGS.

2.4 Experimentation with Protection Envelope

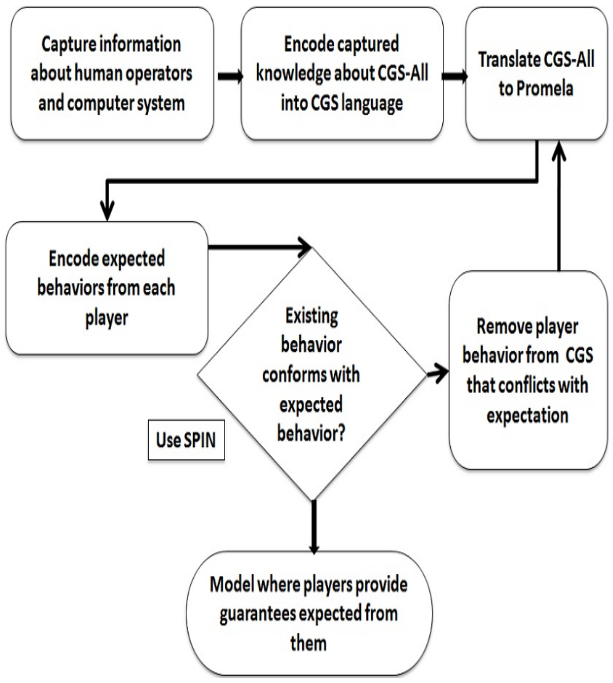
We have shown that given a C_{all}, a player and the player's protection envelope we can verify whether all protected behavior of the player will allow the system to maintain its safety guarantees. The protection envelope generation methodology we have presented this far has the advantage that once a C_{all} is built, using it different protection envelopes can be formulated and analyzed. Every time a new protection property is considered, the "thinned" maximal protected behavior CGS can be generated and verified against safety properties.

2.5 Tutela

We are building a prototype that implements the proposed framework that we call "Tutela". Its functionality in light of the framework presented in Subsection 1.3 is as follows:

2.5.1 Identifying and Modeling Important Players

Tutela provides a GUI to identify the major players in a scenario and build a C_{all} that models their interactions. It allows creation of a repository of the domain knowledge about the various components of a scenario. These include identifying the players of interest, their vocabulary, the guarantees expected of their behaviors, the variables they need to modify to capture the effect of their behavior on the environment. Tutela allows these collections to be dynamically modified at any stage. Tutela assists in building $C_{\mathrm{all}} = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$. Tutela offers a GUI to allow creation and modification of Σ and Π. It also allows creation of the set of variables that are handled by players in C_{all}. The state space of C_{all} can be built in an incremental fashion.



Capture information about human operators and computer system
Encode captured knowledge about CGS-All into CGS language
Translate CGS-All to Promela
Encode expected behaviors from each player
Existing behavior conforms with expected behavior?
Use SPIN
Remove player behavior from CGS that conflicts with expectation
Model where players provide guarantees expected from them

Figure 7: Flow chart for Tutela

Each state needs at least the propositions that will identify it and the enabled vocabulary for each player. Tutela ensures that τ is in the vocabulary of each player in each state. At each state, disallowing a human action indicates domain knowledge on the part of the designer about physical impossibility of that action. A major component of each state is the transitions emanating from that state. We allow new states to be added as needed while creating the set of outgoing transitions from a state. This allows incremental build-up of a CGS. For each state, all possible allowed action combinations need to be handled. Also the variable assignments can be specified at each state. After all required transitions for all states have been specified, the CGS is encoded into an intermediate language which can be translated into the input language for a model checker. The syntax of this CGS specification language is provided at `https://netfiles.uiuc.edu/yasmeen/www/CGSsyntax.html`.

Alternate Method of Creating C_{all} The above interactive method of creating C_{all} can become cumbersome. As mentioned in Section 2.3.4, we can automatically generate C_{all} from a restricted version of C_{all}. Our tool accepts sparse C_{all}s written using the Promela language. Promela is the input language for the model checker SPIN [13]. There are some necessary information that must be present in the incomplete C_{all}: (i) the set of state propositions, (ii) vocabulary of the system and the human operator, (iii) the state space with enabled propositions, (iv) physically impossible human actions, (v) enabled system actions and human actions should be mentioned. Any system action that is not explicitly enabled will be considered to be disabled in the state. Once the enabled action sets for the human and the computer system have been established, the transition system of C_{all} is populated with all specified transitions involving them. Any transition that is not explicitly mentioned in is assumed to lead to an error state. We studied with a simplified self-checkout system scenario where the customer and the checkout system had six actions each. There were six states. Thus there were $6 \times 6 \times 6 = 216$ possible transitions in total. However, in most of the states most of the checkout system actions were physically impossible. Like in the initial state, the checkout system does not accept payments and can not process scanning of an item. Out of the 216 possible transitions 166 were disabled in this scenario. In the initial state, one customer action and four checkout system actions were physically impossible. Thus out of the 36 possible transitions, only 10 were physically possible. Out of the ten possible transitions six led to the error state. Like, attempting to bag an item before pressing the start button. Thus with sparse C_{all} specification method, one only needs to specify only the remaining four meaningful transitions from the initial state instead of all ten possible transitions. An example sparse C_{all} and the complete C_{all} is provided at `https://netfiles.uiuc.edu/yasmeen/www/CGScreation.html`. The Promela encoding of C_{all} is translated into our intermediary language for use in the next steps.

2.5.2 *Modeling Protected Behavior*

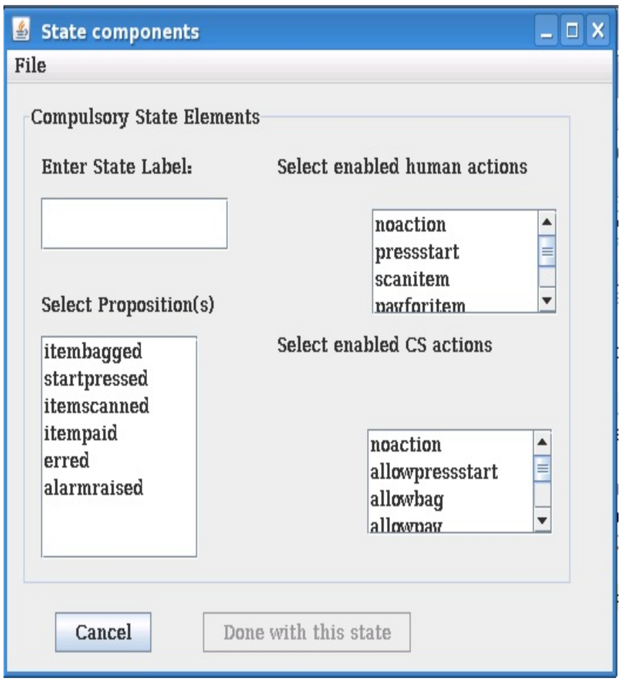


Figure 8: Screenshot from Tutela

Currently one needs to formulate the protection envelope property to Tutela by themselves. In future we intend to augment Tutela with a tool like Propel [29] to have a guided property creation interface. Once C_{all} has been created, we can thin C_{all} with the protection envelope formula to arrive at a model that captures only the protected human behaviors and the computer system's response to them. We can view C_{all} as a finite state automaton where the state space remains the same and the transition function gets modeled by labeled transitions in the automaton. We can perform LTL model checking on this automata to check for protection envelope property satisfaction. Tutela uses the LTL model checker SPIN to help in thinning the C_{all} based on the protection envelopes of the players. C_{all} is thinned by Tutela by implementing an approximation of the definition in Section 2.3.5 by taking the following steps:

- Translate the C_{all} into an automaton and encode the automaton in Promela. The translation occurs as follows: The states are modeled as labeled regions of code. The players are modeled as modules. All state propositions, boolean and integer variables are translated into Promela variables of appropriate type. There is a central "transition controller" module that controls transitions of the automaton. Each player receives the current state from the controller module, makes a non-deterministic choice of action and updates some variables if needed. Depending upon the current state, variable values and action choices made by each player, the controller causes the automaton to transition from one state to another. Hence each state is a combination of labeled regions from the modules of all the players and the controller. The state propositions are always modified by the controller, whereas the player modules are allowed to modify any variable.

- Obtain the Buchii automata for the protection envelope for the player currently under consideration

- Generate scripts to execute SPIN to determine whether the CGS satisfies the protection envelope. Execute those scripts.

- Analyze the output from SPIN to find whether there is a violation of the protection envelope property. If there is a violation, automatically determine the last transition from the counter example generated by SPIN. That transition indicates the action that need to be deleted from a player's available actions. For example, in the self-checkout model we studied, the customer can put an item in the bag in the initial state before even pressing the start button as shown in Figure 5. The proposition indicating that the start button has been pressed is false in the error state and the proposition indicating that an item has been bagged is true in the error state. Thus the error state is a bad or unsafe state. Hence Tutela removes the `BagItem` action from the list of actions enabled for the customer in the initial state in the protection envelope CGS.

- The last two steps are repeated as many times as needed until no counter example trace starting from the initial state can be found.

2.5.3 *Recommended Task Specification Model*

A CGS will be used to internally model the recommended method where the recommended steps dictate the enabled human operator actions in each state. At this point we offer two methods of providing the CGS corresponding to the recommended method. The first one uses the CGS creation GUI to create the CGS corresponding to the recommended behavior, C_{rec}. The other one transforms a sequence of human actions into a CGS. This method requires that the C_{all} be already created. Then each human action in the recommended task execution behavior allows to determine the enabled action function and the transitions starting from the initial state. For example, given a recommended task guidance : `PressStart, ScanItem` and the initial state of of C_{all} as presented in Figure 5, the the initial state of the CGS for the recommended behavior will be as presented in Figure 9.

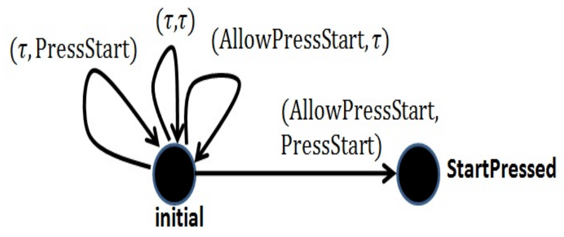


Figure 9: CGS for recommended way of executing task

2.5.4 *Performing Our Framework Steps*

After creating protected behavior CGS C_{PE}, one can verify that it satisfies the safety property via model checking. Then, given a C_{rec} and a C_{PE} Tutela can help determine whether $C_{\mathrm{rec}} \sqsubseteq_{\mathcal{P}} C_{\mathrm{PE}}$ to see if C_{PE} is safe too. Once C_{all} has been created, one can experiment with it by using different protection envelope properties to assess the degree of robustness of the computer system against atypical human operator behavior. In future, we intend for Tutela to (i) automatically assist in assessing the robustness of the computer system against common human errors suggested by Hollnagel [12], like repetition, out of order execution, omission; (ii) automatically further assist in building C_{all}. For example: using user specified boolean propositional restrictions to automatically generate the actions available to players in each state. Given a CGS $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ and a propositional formula φ over propositions in Π for an action a of player p, we translate it into an automaton $C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e', \delta' \rangle$ as follows: for each state $q \in Q$, if $q \models \varphi$ then $a \in e'_p(q)$ otherwise $a \notin e'_p(q)$. δ' is defined as: $\forall q \in Q.\ \forall \bar{v} \in e_{p_1}(q) \times e_{p_2}(q) \times \ldots e_{p_{|P|}}(q).\ \delta'(q, \bar{v}) = \delta(q, \bar{v})$.

3. CONCLUSIONS

We provide a framework for formalizing human operator tasks and determining how resilient the computer system is against human operator action variations. We propose that human operators are unique in their dual nature of being part of a system by behaving expectedly, and being part of the environment by behaving aberrantly. Our framework allows reasoning about the expected behaviors along with controlled study of the manageable aberrant behaviors: protection envelops. We are building a tool for automating the framework. In future we intend to develop a methodology for automatically analyzing the robustness of the protection envelopes with respect to common atypical executions of the recommended procedures.

4. ACKNOWLEDGMENTS

This work was supported in part by NASA Contract number NNA10DE79C and National Science Foundation(NSF) Award number 0917218. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NASA and NSF.

5. REFERENCES

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[2] E. Asarin, O. Maler, A. Pnueli, and J Sifakis. Controller synthesis for timed automata. In *IFAC Symp. System Structure and Control*, pages 469–474. Elsevier, 1998.

[3] Matthew L. Bolton and Ellen J. Bass. Enhanced operator function model: A generic human task behavior modeling language. In *Systems, Man and Cybernetics*, pages 2904–2911. IEEE, 2009.

[4] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[5] Lori A. Clarke, George S. Avrunin, and Leon J. Osterweil. Using software engineering technology to improve the quality of medical processes. In *International Conference on Software Engineering (ICSE 2008)*, pages 889–898, 2008.

[6] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120, 2001.

[7] Dan Diaper. *Task Analysis for Human-Computer Interaction.* Prentice Hall, New Jersey, USA, 1990.

[8] Marlon Dumas and Arthur H. M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In *UML*, pages 76–90, 2001.

[9] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

[10] Elsa L. Gunter, Ayesha Yasmeen, Carl A. Gunter, and Anh Nguyen. Specifying and analyzing workflows for automated identification and data capture. In *HICSS*, 2009.

[11] Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Permissive interfaces. In Michel Wermelinger and Harald Gall, editors, *ESEC/SIGSOFT FSE*, pages 31–40. ACM, 2005.

[12] Erik Hollnagel. The phenotype of erroneous actions. *International Journal of Man-Machine Studies*, 39(1):1–32, 1993.

[13] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley, 2004.

[14] Farnam Jahanian and Aloysius K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Trans. Software Eng.*, 12(9):890–904, 1986.

[15] C.W. Johnson and A.J. Telford. Extending the application of formal methods to analyse human error and system failure during accident investigations. *Software Engineering Journal*, 11(6):335–365, 1996.

[16] Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.*, 5(4):596–619, 1983.

[17] John G. Kemeny. *Report of The President's Commission on the Accident at Three Mile Island: The Need for Change: The Legacy of TMI.* Washington, D.C.: The Commission, 1979.

[18] Orna Kupferman and Moshe Y. Vardi. Robust satisfaction. In *CONCUR*, volume 1664 of *LNCS*, pages 383–398. Springer, 1999.

[19] Frank P. Lees. *Loss Prevention in the Process Industies.* Butterworths, 1980.

[20] Nancy G. Leveson. Software safety: Why, what, and how. *ACM Comput. Surv.*, 18(2):125–163, 1986.

[21] Nancy G. Leveson and Clark Savage Turner. Investigation of the therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[22] Peter A. Lindsay and Simon Connelly. Modelling erroneous operator behaviours for an air-traffic control task. In *Australian User Interface Conference*, pages 43–54, 2002.

[23] C.M. Mitchell. Gt-msocc: A domain for research on human computer interaction and decision aiding in supervisory control systems. *IEEE Transactions on Systems, Man and Cybernetics*, 17(4):553–572, July 1987.

[24] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, New York, NY, USA, 1989. ACM.

[25] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[26] Dongmin Shin, Richard A. Wysk, and Ling Rothrock. Formal model of human material-handling tasks for control of manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 36(4):685–696, 2006.

[27] Rishabh Singh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning component interfaces with may and must abstractions. In *CAV*, volume 6174 of *LNCS*, pages 527–542. Springer, 2010.

[28] A. Prasad Sistla. Safety, liveness and fairness in temporal logic. In *Formal Aspect of Computing*, pages 495–511, 1999.

[29] Rachel L. Smith, George S. Avrunin, Lori A. Clarke, and Leon J. Osterweil. Propel: an approach supporting property elucidation. In *ICSE*, pages 11–21. ACM, 2002.

[30] Juoko Suokas. On the reliability and validity of safety analysis. Technical Report publications 25, Technical Research Center of Finland, Finland, September 1985.

[31] Oksana Tkachuk, Matthew B. Dwyer, and Corina S. Pasareanu. Automated environment generation for software model checking. In *ASE*, pages 116–129. IEEE Computer Society, 2003.

[32] W. M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.

[33] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. Yawl: Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275, 2005.

[34] David J. van Horn. Risk assessment techniques for experimentalists. In *Chemical Process Hazard Review*, pages 23–29, Washington, D.C., 1985. American Chemical Society.

[35] Xiangpeng Zhao, Zongyan Qiu, Chao Cai, and Hongli Yang. A formal model for human workflow. In *International Conference on Web Services (ICWS)*, pages 195–202, 2008.