Evaluating a Specification for its Support of Mode Awareness using Discrete and Continuous Model Checking

Alyssa Byrnes[1] and Cynthia Sturton[1]

***Abstract*— In situations where humans and computers cooperate, mode confusion on the part of the human can be dangerous. We present a methodology to evaluate a semi-autonomous system for its support of mode awareness. The methodology uses discrete-state model checking of the specification and real-valued model checking of the system in operation. Using the ISO standard for adaptive cruise control as a case study, we exhaustively enumerate the instances of four design flaws known to contribute to mode confusion. We then build a real-valued model of eight driving scenarios to find which of those instances of possible mode confusion may lead to a dangerous situation. We find 116 property violations, and determine 62 of them to be potentially dangerous.**

I. Introduction

A semi-autonomous system works in concert with a human operator to achieve a goal. Offloading a portion of the work to the machine can improve safety, reliability, and efficiency compared to a solely human effort. Human–machine cooperation has proven useful in a variety of settings: aviation, industrial automation, nuclear power plants, medical applications, and most recently, automated driving aids. Yet, the problem of automation surprises—when the user is surprised by the behavior of the system—plagues these systems [1], [2], [3], [4], and can lead to unsafe and even fatal conditions [5].

A category of automation surprise is *mode confusion*. A mode defines a set of behaviors of the machine. The functionality, the inputs and outputs available, or the currently controlling entity can all change according to the mode [6], [7]. Mode confusion can occur if the human cannot maintain a valid mental model of the machine's mode or if the system's interface is not clear [2]. These two issues interact: if the system's state evolution is unpredictable, even a perfectly designed interface will be insufficient to prevent mode confusion.

We present a methodology to evaluate the specification[1] of a semi-autonomous system for its support of mode awareness in a dynamic physical environment. The methodology requires only the specification of the semi-autonomous system and examines whether the system evolution can be predictable to the human operator, and if it cannot, what the consequences to safety might be. By focusing on the specification we can consider mode awareness early in the development cycle, before any interface is designed.

[1]Alyssa Byrnes and Cynthia Sturton are with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27514,USA {abyrnes1,csturton}@cs.unc.edu

[1]We use "specification" to refer to the enumerated industry standards of a system.

Prior research has identified numerous categories of design flaw that can lead to automation surprises [6], [2]. We examine four of these categories:[2] *nondeterminism*, *inconsistent behavior*, *operator authority limits*, and *indirect mode changes*. Our insight is that for each category of flaw, a single property can be written that expresses the absence of the category. That is, if the specification is flawed, the associated property will be violated, and an exhaustive enumeration of these violations will find all flaws of a particular category. We can therefore use model checking to exhaustively enumerate all instances of each category of flaw. Unlike prior work (e.g., [8]), our methodology involves a complete exploration of the design, rather than the verification of one transition.

We use UCLID [9], [10], a discrete-state formal verification tool, to look for violations of the four properties in a given specification. We use dReach [11], a formal verification tool that handles nonlinear formulas over real numbers, to determine whether each of the found property violations may lead to unsafe conditions in a physical environment.

As a case study, we examine adaptive cruise control (ACC). Our analysis of the ISO (International Organization for Standardization) standard finds 116 violations of the four properties. We develop eight environment scenarios, and find that of the 116 property violations, 62 could lead to dangerous situations. Our contributions are:

- A methodology that evaluates an English-language specification of a semi-autonomous machine for its support of mode-awareness.
- The formalization of four design flaws known to contribute to mode confusion as properties written in first order logic, and amenable to verification by model checking.
- The application of continuous-valued model checking to prioritize found design flaws for their likelihood to lead to unsafe conditions in a dynamic environment.
- The evaluation of the ACC ISO standard for its support of mode awareness, and the identification of 62 instances where the specification does not support mode awareness and the result may be dangerous on the road.

II. Related Work

The problems that arise when humans and semi-automated systems work together to achieve a goal have been the subject

[2]Different researchers have defined the categories differently. We use the six from Leveson et al. [6], which seem to be comprehensive, and discard the two categories of flaw—unintended side effects and lack of appropriate feedback—that are related to non-mode system state and the interface design, respectively. Those categories are tangential to our central question of whether a system specification supports mode awareness.

of over 30 years of research covering industrial automation, aviation, nuclear power plant process control, and medical applications [3], [2], [1], [12]. The application of formal methods to identify and prevent these problems represents its own subfield, and Bolton et al. provide an excellent review [4]. Here we focus on research specifically targeting mode confusion and adaptive cruise control systems.

A. Mode Confusion

Sarter and Woods define mode awareness as the ability for the user to track and anticipate the behavior of the system [2]. We base the four properties in this paper on the six flaws identified by Leveson et al. [6]. Butler et al. formalize properties related to two of the six flaws for their evaluation of a flight guidance system [8]. They manually identify the preconditions for each mode transition and then verify the transitions occur. The approach is not exhaustive, as there is no guarantee that all such conditions are identified. Joshi et al. do find an exhaustive list of possible design flaws in their analysis of a flight guidance system [13]. However, they use an interactive proof checker which is not fully automated. Others have developed criteria for interface design to mitigate the risk of mode confusion [14], [15], [12]. That work is orthogonal and complementary to our own. A well designed interface coupled with a specification that supports mode awareness will likely yield the best results.

B. Adaptive Cruise Control

Furukawa et al. focus on the interface for adaptive cruise control and conduct user studies to evaluate how well the interface supports mode awareness in complicated driving situations [16]. Horiguchi et al. develop a measure of how likely an adaptive cruise control interface is to cause mode confusion and evaluate their metric with a user study [17]. Lee and Ahn use a state transition table-based approach to look for ambiguities in how the current mode is displayed to the user [18]. Like our work, they base their analysis on the ISO standard.[3] These studies focus on the interface design and are orthogonal and complementary to our work.

C. Cyberphysical Model Checking

O'Kelley et al. use dReach to find ranges of safety for a lane-changing scenario of a car [19]. We use their models as the basis for our eight environment scenarios. Degani et al. present a methodology for formalizing a system design plus a human model that can be reasoned about to find instances where the system state and the human's perceived state do not match and may cause unsafe events [15]. Oishi et al. implement this methodology [20]. Like our work, theirs uses both discrete-state and continuous-valued models. Unlike ours, their work is concerned with finding and defining the boundaries of the dynamic system that lead to automatic mode changes. Bass et al. present a similar methodology using bounded model checking on a human plus system model to find dangerous scenarios [21], but with discrete approximations of the continuous environment.

[3]Their work uses the 2010 version; this work uses the 2018 version.

III. Methodology

Our methodology involves three phases: analyzing the specification, modeling user actions, and assessing the consequences to safety. In the first phase we use discrete-state model checking along with our formalization of standard design flaws to find possible points of mode confusion in the specification. Each such point represents a case where the machine, starting in a state with mode m_m, sensing inputs e from the environment, and receiving inputs u from the user, will transition to mode m'_m, but the user's mental model of the system starting from the same state may transition to one of n possible next modes $m_h^1, m_h^2, \ldots, m_h^n$. This is shown in Figure 1a. The result of this phase is a complete set of tuples (m, u, e) that characterize all such points in the specification.

In the second phase (Figure 1b) we build a transition table that defines what action an ideal user would take for a given *goal*, e.g., to go straight or change lanes, and *environment*—the location and action of the other cars on the road relative to the user's car—when starting in perceived mode m_h^j.

In the last phase (Figure 1c), we use continuous-valued model checking to determine whether the ideal user, in a given scenario (goal plus environment) and taking action in accordance with their perception of the current mode, might end up in a dangerous situation. For each actual state m'_m, each user action u_i, and each scenario the model checker explores how the system will evolve and whether the user's car may crash into an environment car before reaching its goal. We also model the user taking corrective action, albeit delayed. The following sections describe the three phases.

IV. Phase 1: Analyzing the Specification

In phase 1 we find aspects of the specification that could lead to mode confusion.

A. Adaptive Cruise Control (ACC)

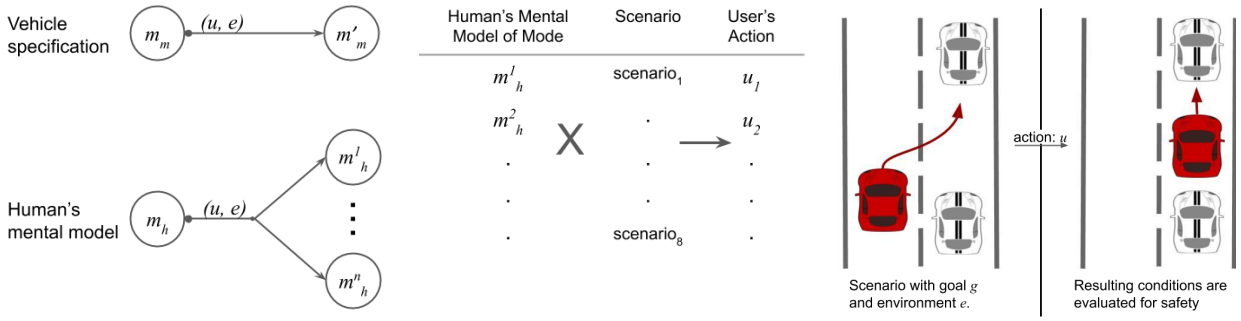
As a case study we target the ACC standard (ISO 15622:2018 [22]). ACC has been in cars internationally since 1998 [23]. With ACC, a car can automatically maintain either a set speed or a set distance from the car in front. We evaluate ACC with Full Speed Range of Automation where the car can come to a complete stop if the car in front stops.

B. Formal Model of ACC Specification

The ACC specification can be modeled as a transition system $\mathcal{M} = (V, m, \Sigma, I, \delta)$ where V is the set of system variables; $m \in V$ is a special variable that indicates the current mode; Σ is the set of events—inputs from the user or the environment—that can cause mode transitions; I is the set of initial valuations to mode m and variables in V; δ is a transition relation between pairs of system states, where each state is characterized by the valuation of mode m and variables in V and transitions are triggered by events in Σ.

A subset of variables $\hat{V} \subseteq V$ are visible to the user. In our model $\hat{V} = \{\text{speed}\}$, the current vehicle speed.

Mode can have one of seven values $m = \{$`Off`, `Standby`, `Following`, `Speed_Control`, `Override`, `Error`$\}$. The two *active* modes of operation are `Following` and



(a) Phase 1 (b) Phase 2 (c) Phase 3

Fig. 1: Overview of methodology. In phase 1, model checking enumerates the property violations in the specification. In phase 2, a transition table defines the human model. In phase 3, For every next-state mode of the machine m'_m, for every scenario, user action u is taken and the resulting situation is evaluated for safety.

`Speed_Control`. In `Following`, the car's speed is determined by the distance from the car in front and in `Speed_Control` the car's speed is determined by the speed set by the driver. The car switches between the two modes based on which speed is lower. The car enters `Hold` when it is in `Following` and the car in front comes to a complete stop. At this point, a timer starts. If the car in front does not move in a specific amount of time, the car exits `Hold` and goes to `Standby`. If the car in front does move within the specified time, the car returns to `Following`. Lastly, when the car is in either `Speed_Control` or `Following` and the driver presses the gas, the car transitions to `Override`.

The events $\Sigma = \{(u, e) : u \in U, e \in E\}$ are tuples of user inputs (U) and environmental inputs (E).

The transition relation δ is detailed in Table I. Some transitions were ambiguous; in that case the model includes all possible transitions. For example, the specification states that the system can transition from `Hold` mode to an active ACC mode automatically, but does not give the transition conditions nor specifies which ACC active mode. The model defines it as a nondeterministic transition to either `Following` or `Speed_Control`.

C. Properties for Finding Design Flaws

We consider four properties: Operator Authority (OA), Consistent Behavior (CB), Determinism (Det), and Direct Mode Changes Only (DMCO). Each property formalizes desired behavior, i.e., the absence of the associated design flaw. There is a partial ordering to the properties: OA $\Rightarrow$ CB $\Rightarrow$ Det. That is, for any model where Operator Authority is satisfied, Consistent Behavior is satisfied; and for any model where Consistent Behavior is satisfied, Determinism is satisfied. There is no strict ordering of Direct Mode Changes Only with the other three. Among the first three properties, Operator Authority is the strongest. If OA is true of the system, it might be reasonable to design an interface that can handle the burden of making the system usable. Conversely, a violation of Determinism, which is the weakest of the ordered three, would indicate a confusing system; it is unlikely that any interface could make the system usable.

The three ordered properties are related to nondeterminism: each considers whether mode transitions behave predictably for varying levels of fixed internal and environmental state. The more information the user is required to track in order for the mode to be predictable, the harder it will be for the user to maintain mode awareness.

Like all properties related to nondeterminism, these three properties are hyperproperties, rather than trace properties, and cannot be verified using standard model checking techniques [24]. However, they are amenable to verification by self-composition [25], [26]. To formally express these properties, we create two instances of the vehicle model: $\mathcal{M}_1 = (V_1, m_1, \Sigma_1, I_1, \delta_1)$ and $\mathcal{M}_2 = (V_2, m_2, \Sigma_2, I_2, \delta_2)$ with events $\Sigma_1 = \{U_1, E_1\}$ and $\Sigma_2 = \{U_2, E_2\}$. The properties are formally stated and evaluated over the asynchronous composition of the two instances. Direct Mode Changes Only is a trace property and can be verified by standard model checking. We describe each of the properties in turn. We use the notation m' to indicate the next mode after a transition and $\mathbf{v}$ to indicate the vector of all variables in V.

Property 1 (Direct Mode Changes Only):

$$(u_1 = nil) \rightarrow (m_1 = m'_1).$$

DMCO states if the user input is nil the mode m does not change. If DMCO is violated mode transitions can happen without any user action. An ideal user with a perfect understanding of the specification has to mentally track the value of all internal system variables (e.g., time-out timers) and notice all changes environment changes to correctly track the current mode and predict the next mode change.

Property 2 (Determinism):

$$(m_1 = m_2) \wedge (u_1 = u_2) \wedge (e_1 = e_2) \wedge (\mathbf{v}_1 = \mathbf{v}_2) \rightarrow (m'_1 = m'_2).$$

Determinism says the next mode is determined by current mode, current variable valuations, user inputs, and environment inputs. A violation of Determinism means mode transitions are unpredictable. Even with perfect knowledge of the specification, the user cannot predict mode changes and must rely on the interface to know their current mode.

TABLE I: Transitions of ACC. The last column gives the location of the definition in the specification.

Start Mode	Next Mode	Transition Condition	Specification
`Off`	`Standby`	"ACC Active" button pressed	6.1
Any mode - `Off`	`Off`	"ACC Active" button unpressed	6.1
`Standby`	`Following`, `Speed_Control`, `Hold`	"ACC" button pressed	6.1
`Following`, `Speed_Control`, `Hold`	`Error`	error occurs	6.6
	`Override`	driver presses gas	6.3.1.4
	`Standby`	"ACC" button pressed Driver presses brake	6.1 6.3.1.2
`Error`	`Off`	"ACC Active" button pressed	6.6
`Following`	`Hold`	car stopped for some t $<$ 3 sec	6.1
	`Speed_Control`	set speed $<$ following speed	6.1
`Speed_Control`	`Following`	following speed $<$ set speed	6.1
`Override`	`Following`, `Speed_Control`, `Hold`	driver releases gas	6.3.1.4
`Hold`	`Following`, `Speed_Control`	automatic/random driver request	6.2.4 6.2.4

Property 3 (Consistent Behavior):

$$(m_1 = m_2) \wedge (u_1 = u_2) \wedge (e_1 = e_2) \wedge (\hat{\mathbf{v}}_1 = \hat{\mathbf{v}}_2) \rightarrow (m_1' = m_2').$$

CB says that mode transitions depend on current mode, current valuations to user-visible variables, user inputs, and environment inputs. If CB is violated the same driver input in the same starting mode can cause a transition to different modes depending on the state of hidden system variables. An ideal user with a perfect understanding of the specification would have to track the current value of internal system variables and notice all changes to the environment.

Property 4 (Operator Authority):

$$(m_1 = m_2) \wedge (u_1 \neq nil) \wedge (u_2 \neq nil) \wedge (u_1 = u_2) \rightarrow (m_1' = m_2').$$

OA says that mode transitions depend on current mode and user inputs. If OA is violated the driver's input may be ignored depending on the state of hidden system variables or inputs from the environment. An ideal user with perfect understanding of the specification would have to mentally track the current value of all internal system variables and the state of the environment. Absent that ability, the user would have to rely on the interface to understand how or if their input affected the current mode.

V. Phase 2: Modeling User Actions

In phase 2 we model the user's actions. The model is a transition table (see Tables II and III) that defines the user's action given their goal (to go straight or change lanes), their environment, and the perceived mode. The table is used in phase 3 to determine user actions for a given scenario.

VI. Phase 3: Assessing Consequences to Safety

In phase 3 we identify those instances of possible mode confusion that could cause accidents. For each scenario (goal plus environment), perceived mode m_m', and user action m_h^j,

TABLE II: Human Model When Goal is to Go Straight

Enviroment	Perceived Mode	Action
Front Car Decels	`Off`	Brake
	`Standby`	Brake
	`Following`	Nothing
	`Speed_Control`	Brake
	`Override`	Brake
	`Hold`	NA
	`Error`	Brake
Front Car Accels	`Off`	Press Gas Pedal
	`Standby`	Press Gas Pedal
	`Following`	Nothing
	`Speed_Control`	Press Gas Pedal
	`Override`	Press Gas Pedal
	`Hold`	Nothing
	`Error`	Press Gas Pedal

we examine whether the user's car might crash into one of the other cars on the road.

A. Scenarios

Each scenario involves three cars: the user's car plus two other cars on the road. The user has one of two goals: continue forward or change lanes. The other cars are either accelerating or decelerating. Table IV lists the scenarios.

B. Modeling Physical Dynamics

The safety assessment uses a model that includes the physical dynamics of the vehicles and the human actions and reactions for each possible point of confusion. The model is defined by the tuple $\mathcal{C} = (V_u, V_f, V_b, I, T, E, H)$ where V_u, V_f, V_b represent the user's vehicle, front vehicle, and back vehicles respectively. I represents the initial state of the vehicles, T represents the goal, E represents the environment, and H models the human.

In keeping with prior work [19], the vehicles are represented by a parameterized, nonlinear, 7 degree of freedom 3-D bicycle model [27], $V = (B, \Psi, \dot{\Psi}, v, x, y, \delta)$, where B

TABLE III: Human Model When Goal is to Change Lanes

Enviroment	Perceived Mode	Action
Front Car Decels	`Off`	Release Gas Pedal
	`Standby`	Release Gas Pedal
	`Following`	Brake
	`Speed_Control`	Brake
	`Override`	Release Gas Pedal
	`Hold`	NA
	`Error`	Release Gas Pedal
Front Car Accels	`Off`	Press Gas Pedal
	`Standby`	Press Gas Pedal
	`Following`	Press Gas Pedal
	`Speed_Control`	Nothing
	`Override`	Press Gas Pedal
	`Hold`	Press Gas Pedal
	`Error`	Press Gas Pedal

TABLE IV: Scenarios Tested

Scenario #	Goal	Front Car Action	Back Car Action
1	Go Straight	Decel	Decel
2			Accel
3		Accel	Decel
4			Accel
5	Change Lanes	Decel	Decel
6			Accel
7		Accel	Decel
8			Accel

is slip angle, Ψ is heading angle, $\dot{\Psi}$ is yaw rate, v is velocity, x and y are the coordinate positions, and δ is the angle of the front wheel. The vehicle location is given as a function of time by the set of ordinary differential equations of the vehicle model as defined by Althoff et al. [28].

The initial state I determines the initial values for each parameter in V_u, V_f, and V_b, except Ψ. The trajectory T determines $\Psi = v_d * \kappa_d$, where v_d is the desired velocity and κ_d is the desired curvature. The desired curvature is described by a cubic spline. We use the trajectory planner from O'Kelley et al. to define the curvature of our vehicle [19]. The environment E determines the rate of acceleration or deceleration for each vehicle. Finally, the human model H defines the user's initial action—as determined by the perceived mode and the transition tables (Tables II, III)—plus the user's corrective action once they realize their mistake, and the time delay between the two actions.

C. Safety Analysis

For a given instance of the tuple $\mathcal{C}$ modeling a possible point of confusion, continuous-valued model checking is used to determine for which initial values I would the user's car crash into, i.e., have overlapping (x, y) coordinates with, either of the environment vehicles given the trajectory, environment, and human model.

The set of initial values I which do not lead to a crash are the *safety set*. Let c be an instance of the tuple $\mathcal{C}$ in which the user is not confused, and let c' be a second instance, identical to the first except that the user is confused and takes an initial action followed by delay and then corrective action. If the safety set for c' is equal to the safety set for c we say the point of confusion modeled by c' is likely not dangerous. If, on the other hand, the safety set for c' is a subset of the safety set for c, then we say the point of confusion is dangerous.

$$\text{if } \mathit{safetyset}(c') = \mathit{safetyset}(c) \text{ then safe}$$
$$\text{if } \mathit{safetyset}(c') \subset \mathit{safetyset}(c) \text{ then unsafe}$$

VII. Implementation

A. Phase 1: Analyzing the Specification

We implement the ACC formal model in UCLID, an automated verification tool [9], [10]. UCLID's modeling language is both declarative and imperative and includes support for the theories of linear integer arithmetic, equality, arrays, bitvectors, and uninterpreted functions. The ACC model made use only of the first two, but the additional theory support would allow UCLID to handle more complex specifications.

Each of the four properties were expressed in UCLID's specification language, which supports a decidable fragment of quantifier-free first order logic. For the three properties related to nondeterminism (OA, CD, and Det), we used self-composition [25], [26] to find violations of the property: the model comprises two instances of ACC, and the property was evaluated over the composed two-instance model.

We used UCLID to exhaustively find all violations of each property. To begin, UCLID is given the model $\mathcal{M}$ and property p, and returns an interpretation to the model I such that I violates the property: $I \not\models p$. In the next iteration, UCLID is given the model $\mathcal{M}$ and a new property $p \vee I$ and returns an interpretation to the model I^1 such that $I^1 \not\models p \vee I$. The exploration continues until either UCLID finds that the model is satisfied by the newest property: $\mathcal{M} \models p \vee I \vee I^1 \vee \ldots \vee I^m$ or all interpretations have been considered. The exploration is guaranteed to terminate, as the model is finite and therefore there exists a finite number of interpretations. The iterative exploration is implemented as a Python wrapper to UCLID.

B. Phase 2: Modeling User Actions

In phase 2 the human's action is defined for each possible goal, environment, and perceived mode. The human model is implemented in Python as a transition table. Phase 3 uses this Python model when building each verification instance.

C. Phase 3: Assessing Consequences to Safety

We use model checking tool, dReach, and its backend solver, dReal [11], to implement the model of the physical dynamics of the vehicles and the human actions and reactions. Each instance in which the user thinks they are in mode m'_h and takes action u, when in fact they are in mode m'_m, is evaluated for each of the eight scenarios. The evaluation determines whether the user's car will reach the goal without crashing into either of the environment cars. The dReal solver is a decision procedure for nonlinear formulas over real numbers. Given a system of equations, it returns either δ-*satisfiable*, meaning the system of equations are satisfiable

within some range of error δ, or unsatisfiable if there is no satisfying solution to the equations. We use the front end, dReach, to describe each scenario: the system of differential equations that describes each vehicle's movement, the change in acceleration and trajectory over time given the human's actions, and the goal of the user's vehicle.

The vehicle parameters and controller parameters were set according to the values determined empirically by Snider [29]. We set the starting distance between the user's vehicle and environment vehicles at 4.5 meters. The vehicle speed can range from 0 $\mathrm{m/s}$ to 40 $\mathrm{m/s}$. The maximum acceleration and deceleration are 2.87 $\mathrm{m/s^2}$ and 4.33 $\mathrm{m/s^2}$ respectively, as determined by Bokare and Maurya experimentally [30].

When the user is confused about the current mode, and takes action accordingly, they will always (in our model) realize their confusion and take corrective action after some delay. Using experimentally shown responses in adaptive cruise control scenarios [31], we model the delay at 1.5 s.

VIII. Discussion of Results

We found 116 property violations in the ACC specification. As shown in Table VI, 46 of these violated Determinism, 38 violated Consistent Behavior, 8 violated Operator Authority, and 24 violated Direct Mode Changes Only. Of the 116 violations, 62 were found to be dangerous; these dangerous property violations are described in Table V.

We did not double count violations, but all properties that violate a weaker determinism property also violate the stronger property. For example, all violations of Determinism could be included as a violation of Operator Authority.

We traced these properties to seven *ambiguities* in the ISO specification. Each ambiguity causes at least one of the 62 dangerous property violations. Table VII provides the details and Figure 2 shows the distribution of each ambiguity causing violations of the properties.

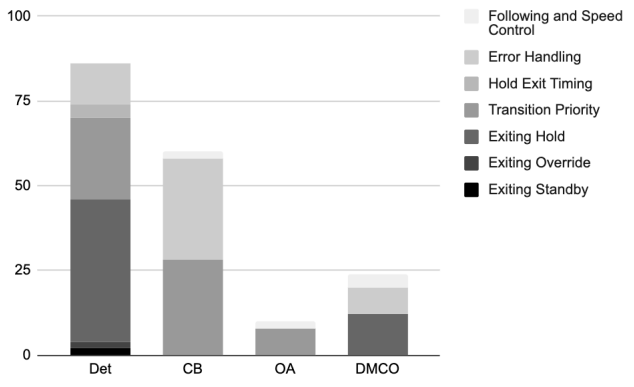


Fig. 2: Type of Ambiguity Associated with Each Property

Ambiguity 1 (Exiting Standby): Exiting `Standby` transitions ACC to an active mode, but the specification does not clarify which one. The car manufacturer is free to choose either `Speed_Control` or `Following`. We suggest strengthening the specification to state that ACC always goes to `Speed_Control` first.

Ambiguity 2 (Exiting Override): Similarly, exiting `Override` transitions ACC to an active mode, but the specification does not clarify which one. We suggest strengthening the specification to state that ACC returns to its prior mode when exiting `Override`.

Ambiguity 3 (Exiting Hold): The ACC specification allows for automatic transitions from `Hold` mode to any other mode, but does not specify under which conditions those transitions may occur nor what happens when the user manually exits `Hold`. We suggest that all automatic transitions from `Hold` be fully specified and when the driver manually exits `Hold`, ACC should transition to `Standby`.

Ambiguity 4 (Transition Priority): If multiple events happen, it is unclear which takes priority. This can lead to 32 potentially dangerous transitions. For example, if the car is in `Speed_Control` mode, and the car in front slows when the operator presses the cancel button, the specification does not state whether the car should transition to `Following` because of the reduced speed or `Standby` because of the user input. We propose that system errors are given the highest priority, user inputs the second highest, and internal commands and environmental inputs the lowest priority.

Ambiguity 5 (Hold Exit Timing): ACC transitions from `Hold` to `Standby` after a set amount of time elapses. Without knowledge of the internal timer, the driver could think they are in `Hold` when they are in `Standby`, think they are in `Following` when they are in `Standby`, or think they are in `Standby` when they are in `Following`. We suggest this ambiguity can be handled by the interface, which can communicate when the transitions happen.

Ambiguity 6 (Error Handling): The user may not know when the system transitions to `Error` mode. This ambiguity led to 11 potentially dangerous transitions. One example is the transition from `Following` to `Error` mode. The user, expecting the braking and acceleration behavior from `Following` mode, may not be prepared to act when needed. We suggest this ambiguity be handled by the interface.

Ambiguity 7 (Following and Speed Control): ACC switches between `Following` and `Speed_Control` automatically. If the switch is not made clear, the user may not know which mode they are in. This ambiguity led to four potentially dangerous transitions. We suggest this ambiguity be handled by the interface.

IX. Conclusion

We analyze the ISO standard for adaptive cruise control for its support of mode awareness. We exhaustively enumerate violations of four properties known to support mode awareness and leverage those findings to discover seven ambiguities in the specification. Our safety analysis demonstrates each ambiguity to be potentially dangerous.

X. Acknowledgments

We thank the anonymous reviewers and Sanjit Seshia for the helpful feedback. This research is supported by the National Science Foundation under the Cyber-Physical Systems Frontier project, VeHiCaL: Verified Human Interfaces,

TABLE V: Dangerous Transitions for Each Scenario

expected next-mode	actual next-mode	dangerous scenario
`Following`	{`Speed_Control`, `Off`, `Standby`, `Error`}	1, 2, 3, 4
`Following`	`Speed_Control`	1, 2, 3, 4, 5, 6
`Hold`	{`Speed_Control`, `Off`, `Standby`, `Error`}	3, 4
`Speed_Control`	`Following`	5, 6, 7, 8
{`Standby`, `Off`, `Error`, `Override`}	{`Following`, `Speed_Control`}	5, 6
`Speed_Control`	{`Standby`, `Hold`, `Off`, `Error`, `Override`}	7, 8

TABLE VI: Count of Dangerous Violations per Property

Property	# Violations	# Dangerous Viol.
Nondeterminism	46	24
Inconsistent Behavior	38	12
Lack of Operator Authority	8	8
Indirect Mode Change	24	18
Total	116	62

TABLE VII: Count of Dangerous Violations per Ambiguity

Ambiguity	# Violations	# Dangerous Viol.
Exiting Standby	2	2
Exiting Override	2	2
Exiting Hold	54	25
Transition Priority	60	25
Hold Exit Timing	4	3
Error Handling	50	11
Follow or Spd Control	4	4

Control, and Learning for Semi-Autonomous Systems (grant No. CNS-1544924). Any opinions, findings, conclusions, and recommendations expressed in this paper are solely those of the authors.

REFERENCES

[1] N. B. Sarter, D. D. Woods, C. E. Billings, *et al.*, "Automation surprises," *Handbook of human factors and ergonomics*, vol. 2, 1997.

[2] N. B. Sarter and D. D. Woods, "How in the world did we ever get into that mode? mode error and awareness in supervisory control," *Human Factors*, vol. 37, 1995.

[3] D. A. Norman, "The problem with automation: inappropriate feedback and interaction, not over-automation," *Philosophical Trans. of the Royal Society B*, 1990.

[4] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using formal verification to evaluate human-automation interaction: A review," *Trans. on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 43, 2013.

[5] C. E. Billings, *Aviation automation: The search for a human-centered approach*. Mahwah, NJ: Lawrence Erlbaum Associates, 2018.

[6] N. Leveson, L. D. Pinnel, S. D. Sandys, S. Koga, and J. D. Reese, "Analyzing software specifications for mode confusion potential," in *Workshop on Human Error and System Development*. Glasgow Accident Analysis Group, 1997.

[7] A. Degani, "Modeling human-machine systems: On modes, error, and patterns of interaction," Ph.D. dissertation, Georgia Institute of Technology, 1996.

[8] R. W. Butler, S. P. Miller, J. N. Potts, and V. A. Carreno, "A formal methods approach to the analysis of mode confusion," in *17th Digital Avionics Systems Conf.*, vol. 1. AIAA/IEEE/SAE, 1998.

[9] R. E. Bryant, S. K. Lahiri, and S. A. Seshia, "Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions," ser. LNCS 2404, E. Brinksma and K. G. Larsen, Eds., 2002.

[10] S. A. Seshia and P. Subramanyan, "UCLID5: Integrating modeling, verification, synthesis, and learning," in *MEMOCODE*. IEEE, 2018.

[11] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *International Conf. on automated deduction*. Springer, 2013.

[12] G. A. Jamieson and K. J. Vicente, "Designing effective human-automation-plant interfaces: A control-theoretic perspective," *Human Factors*, vol. 47, 2005.

[13] Joshi, Miller, and Heimdahl, "Mode confusion analysis of a flight guidance system using formal methods," in *22nd Digital Avionics Systems Conf.* IEEE, 2004.

[14] M. Heymann and A. Degani, "Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm," *Human Factors*, vol. 49, 2007.

[15] "Formal Verification of Human-Automation Interaction," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 44, 2002.

[16] H. Furukawa, T. Inagaki, Y. Shiraishi, and T. Watanabe, "Mode awareness of a dual-mode adaptive cruise control system," in *International Conf. on Systems, Man and Cybernetics*, vol. 1. IEEE, 2003.

[17] Y. Horiguchi, R. Fukuju, and T. Sawaragi, "An estimation method of possible mode confusion in human work with automated control systems," in *SICE-ICASE International Joint Conf.* IEEE, 2006.

[18] S. H. Lee and D. R. Ahn, "Design and verification of driver interfaces for adaptive cruise control systems," *Journal of Mechanical Science and Technology*, vol. 29, 2015.

[19] M. O'Kelly, H. Abbas, S. Gao, S. Shiraishi, S. Kato, and R. Mangharam, "APEX: Autonomous vehicle plan verification and execution," 2016.

[20] M. Oishi, I. Mitchell, A. Bayen, C. Tomlin, and A. Degani, "Hybrid verification of an interface for an automatic landing," in *Conf. on Decision and Control*, vol. 2. IEEE, 2002.

[21] E. J. Bass, K. M. Feigh, E. Gunter, and J. M. Rushby, "Formal modeling and analysis for interactive hybrid systems," *Electronic Communications of the EASST*, vol. 45, 2011.

[22] I. 15622, "Intelligent transport systems–adaptive cruise control systems–performance requirements and test procedures," 2018.

[23] P. Bhatia, "Vehicle technologies to improve performance and safety," 2003.

[24] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, 2010.

[25] G. Barthe, P. R. D'Argenio, and T. Rezk, "Secure information flow by self-composition," in *Computer Security Foundations Workshop*. IEEE, 2004.

[26] R. Shemer, A. Gurfinkel, S. Shoham, and Y. Vizel, "Property directed self composition," in *International Conf. on Computer Aided Verification*. Springer, 2019.

[27] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[28] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *Trans. on Robotics*, vol. 30, 2014.

[29] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[30] P. Bokare and A. Maurya, "Acceleration-deceleration behaviour of various vehicle types," *Transportation research procedia*, vol. 25, 2017.

[31] K. Zeeb, A. Buchner, and M. Schrauf, "Is take-over time all that matters? The impact of visual-cognitive load on driver take-over quality after conditionally automated driving," *Accident Analysis & Prevention*, vol. 92, 2016.