FORMAL METHODS WITH DYNAMIC AGENT SAFETY LOGIC

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

Seth Ahrenbach

Dr. Rohit Chadha, Thesis Supervisor

MAY 2005

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

THE TITLE OF YOUR
PROJECT GOES HERE

presented by Your Name,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Advisor Name

Dr. Committee Member

Dr. Committee Member

Dr. Committee Member

Dr. Committee Member

ACKNOWLEDGMENTS

This page is where you would acknowledge all those who helped you with your academic research.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

This is the abstract of your dissertation project. It should not exceed one page.

Chapter 1

Introduction

In this doctoral thesis, I present a logic for reasoning about safety-critical information flow among machines and humans. The thesis advances the domain of modal logic by developing a rich and expressive logic suitable for reasoning about real humans in real situations, which in turn provides a new tool for formal methods researchers interested in developing safe human-machine hybrid systems. Thus, the thesis is quite interdisciplinary, pulling from fields as diverse as philosophy, game theory, computer science, and safety engineering.

I have developed a logic, which I call Dynamic Agent Safety Logic (DASL). It is based on the foundations of game theory, in which models of agency formally capture how knowledge, rationality, and action relate. Game theory presents a model that, given a description of a scenario, allows one to deduce what actions are dictated by a given theory of rationality. The standard game-theoretic inference works as follows:

$$Knowledge\_of\_Situation \wedge Rationality \Rightarrow Good\_Action.$$

However, the foundational assumptions of game theory do not accurately capture real human reasoning, and as a result humans frequently deviate from the prescribed

behavior. Game theory offers normative analysis of behavior, not descriptive. Looking at the above formula, we can ask a question: what can we infer when an agent fails to execute the prescribed action, as when pilots provide unsafe control inputs to their aircrafts? We can answer this question by examining the contrapositive of the above game-theoretic inference:

$$\neg Good\_Action \Rightarrow \neg(Knowledge\_of\_Situation \wedge Rationality),$$

or equivalently,

$$\neg Good\_Action \Rightarrow \neg Knowledge\_of\_Situation \vee \neg Rationality.$$

With a bit more Boolean manipulation, we have the following:

$$\neg Good\_Action \wedge Rationality \Rightarrow \neg Knowledge\_of\_Situation.$$

Thus, embedded in the classical game-theoretic model of agency is a logical inference from bad action to missing knowledge, assuming the agent is rational. This makes intuitive sense upon reflection. If someone is rational, yet they commit an irrational (read: "bad") action, then it must be the case that they didn't know some crucial information. With this insight in hand, my goal was to identify the logic in which the above inference is sound, ideally with details about which particular pieces of information are missing from an agent's knowledge base when she executes a bad action. Again, it should not be surprising that such a logic exists, because classical game theory already posits a *logical* relationship between knowledge of particular propositions and particular actions.

With DASL, I have formally captured such inferences, where a rational agent executes a bad action, and from this we can infer which safety-critical information

they are missing. I apply this technique to aviation safety as a formal method, but in principle I believe it could be applied to many domains of human agency that meet certain conditions.

My research validates the approach by using DASL to analyze aviation mishaps, illustrating its usefulness. My proposal is to extend this research by constructing a monitor prototype based on the logic suitable for runtime diagnosis of information misflow, that is, when safety-critical information fails to reach the human agent and inform her actions. Because we can deduce which safety-critical information is missing from her knowledge base, we can automatically act to correct this misflow.

This is where information assurance comes in. Information assurance is the field of computer science studying the desirable properties of information systems relating to information flow. In particular, information assurance is concerned with properties like confidentiality, integrity, and availability, among others. If a system is designed to interact with a human, then one of its desired properties is that the safety-critical information successfully flows to the human and informs her actions. Sometimes humans become overwhelmed by information competing for their attention, especially during emergency situations. This phenomenon is called information overload. It leads to human behavior that is suboptimal and often dangerous [45]. The problem, in terms of information assurance properties, is that some safety-critical information is not reaching the human component because the human component's cognitive resources are unavailable, suffering from a sort of denial-of-service attack. I propose to formally characterize this situation in my research, and offer strategies for automated enforcement of safety-critical information reaching the human component.

In what follows, I will describe the relevant background material in Section **??**, including the foundations of game theory and the logical models of agency informing my developments. In Section **??**, I present the logic DASL, and prove that it is sound. In Section **??**, I illustrate its application to an aviation mishap, formalized in the Coq

Proof Assistant for added formal rigor. In Section **??**, I discuss the algorithm that can leverage the logic's power to deduce missing information from flawed action, and describe how my next steps will be to implement it as a prototype.

Chapter 2

Background

This chapter describes the context in which my research makes advances. Section 2.1 describes the state of formal methods. Section 2.3 introduces the logic to be used in exploiting this opportunity, a variation of dynamic epistemic logic modified for reasoning about pilots. Section 2.3 describes related work involving logical approaches to agents in system verification. Section 2.4 describes the Coq Proof Assistant, a tool for constructing mechanically checked mathematical proofs, to be used in this research.

2.1 Formal Methods

Formal methods increase confidence that hardware and software components function correctly [17]. They involve the application of mathematical techniques to the design and analysis of systems, usually the safety-critical components [15]. A standard approach is to develop an abstract specification of the software or hardware system, design the system to meet those specifications, and then use formal logic to prove that the designed system meets the desired safety specifications.

According to the Federal Aviation Administration (FAA), one of the top 12 causes of mistakes in the aviation workplace is a lack of awareness [1]. According to Boeing and the FAA, approximately 80 per cent of aviation accidents (including maintenance accidents) are due to human error [2, 3]. To increase safety in human factors, the industry relies on education, psychology, anthropometrics, safety engineering, and to a limited extent, computer science. The primary focus of computer science research regarding human factors is the design and testing of software systems that are easy and intuitive for humans to interact with.

Some researchers have sought to develop formal methods for mitigating human-induced sources of failure [14, 21]. Thus far this work has focused on the development of formal methods tools for analyzing human machine interaction during the design and specification phase, rather than at runtime. The goals have been to develop software, and techniques for verifying the correctness of that software, to avoid mode confusion, a type of pilot error wherein the pilot believes the autopilot is in one mode, when in reality it is in another. In these situations, the autopilot is not offering protections concerning flight control inputs like thrust and pitch, so the pilot risks providing dangerous inputs.

Butler, Miller, Potts, and Carreno [14] trace mode confusion to three sources:

1. poor display of automation state
2. unnecessarily complex automation
3. flight crew has an incorrect mental model of the state of the aircraft

They say human factors research focuses on mitigating item (1) above, and they develop formal models in the PVS automated theorem prover to address items (2) and (3). However, the formal models they develop are of the automation system, not of the human components.

Similarly, Rushby [21] describes a class of errors he calls automation surprises, which are distinct but related to mode confusion. These occur when a pilot becomes surprised by the automated behavior of the system. He proposes a formal method addressing automation surprises that constructs a model of the system behavior, constructs a formal specification of a possible pilot's mental model of the system, and compares them for disagreement. His solution focuses on identifying potential automation surprises in the system, and both models are of the system behavior itself: one directly of the model, and one indirectly of the model, mediated through a hypothetical pilot's mind.

In each of the above cases, the formal efforts focus on modeling the system itself, rather than on the human component, and they address the problem at design time. This work complements theirs by constructing a formal model of the pilot herself, and addressing the problem at runtime. Similarly, this work aims to address a variety of problems due to a pilot's loss of situational awareness, including mode confusion. The next section provides background information on the mathematical models on which I base my own.

2.2 Game Theory

Game theory is a mathematical model for strategic reasoning. Strategic reasoning refers to the way an agent reasons in situations where her payoffs depend on the actions of other agents in addition to her own, and in which she knows about these dependencies. For turn-based games, the mathematical structure employed is a *game tree*, where each node represents a player's turn, and each edge the transition via a player's action. The leaves of the tree represent the payoffs each player receives at the end of the game. This paper is not concerned with the games themselves, but rather with the underlying assumptions about agency that entail their solutions. We

briefly illustrate these underlying assumptions with the following example.

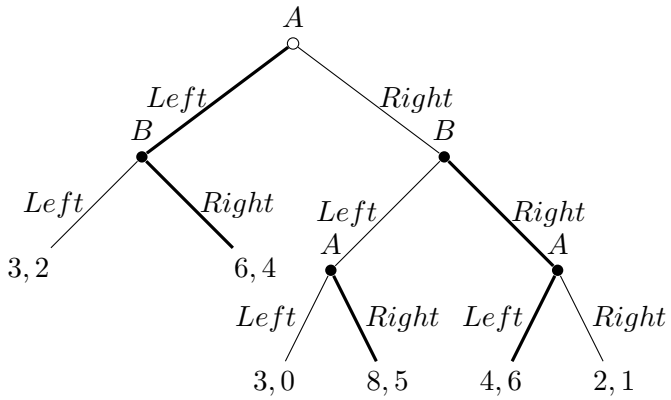


Figure 2.1: A game between players A and B

We can see in the figure below that the first player to act is Player A, at the root node. Her choices are to move *Left* or *Right*. Player B faces similar choices at the resulting nodes, and the players alternate turns until the game ends and they receive their payoffs, listed *(A,B)*. Briefly glancing through the outcomes, it looks like A should aim for the node with payoff *(8,5)*, because 8 is the highest payoff available to A. However, the official solution to the game is for A to first go *Left*, and then for B to go *Right*, resulting in a payoff of *(6,4)*, where both get less than the intuitively appealing outcome! Why is this so?

Game theory makes strong assumptions about agent knowledge and rationality. The solution to this game is reached through an algorithm called *backward induction* [20]. The players reason by starting at each end node and looking to the immediate parent node, and asking what the deciding player will do at that node, assuming she will choose the path with the highest payoff. So, at the bottom right of the figure, player A is to act, and she can go *Left* for a payoff of 4, or *Right* for a payoff of 2. So, she will go *Left*, illustrated by the bold line. The end nodes not selected are subsequently eliminated. This process is repeated at each end node. Then it is recursively applied up the tree. So along the right branch, player B decides between

Left for a payoff of 5, or *Right* for a payoff of 6, because B knows that A is rational, and he knows how she will act at each end node. A, at the root, then must choose between *Left* for a payoff of 6, or *Right* for a payoff of 4, because she knows that B is rational, and knows that B knows that she is rational. The explanation begins to illustrate the assumptions game theory makes about each player's knowledge. In fact, this only scratches the surface.

Game theory, and classical economics in general, makes the following assumptions about agent knowledge, formalized in epistemic logic [44].

Agency Model in Classical Game Theory.

(1) $\mathbf{K_i}\,(\varphi \Rightarrow \psi) \Rightarrow (\mathbf{K_i}\,\varphi \Rightarrow \mathbf{K_i}\,\psi)$

(2) $\mathbf{K_i}\,\varphi \Rightarrow \varphi$

(3) $\mathbf{K_i}\,\varphi \Rightarrow \mathbf{K_i}\,\mathbf{K_i}\,\varphi$

(4) $\neg\mathbf{K_i}\,\varphi \Rightarrow \mathbf{K_i}\,\neg\mathbf{K_i}\,\varphi$

(5) $\mathbf{C}_G((1) \wedge (2) \wedge (3) \wedge (4) \wedge (5))$.

This forms an idealized model of the knowledge component of classical game theory's agents. $\mathbf{K_i}$ is a modal operator for knowledge, and $\mathbf{K_i}\,\varphi$ reads, "agent i knows that φ." $\mathbf{C}_G$ is a modal operator for common knowledge, the fixpoint for "everyone in group G knows that everyone knows that..." The agents are logically omniscient due to (1), knowledge implies truth with (2), agents have *positive introspection* with (3), and *negative introspection* with (4). Assumptions (1), (3), (4), and (5) are somewhat dubious. The model also fails to formally represent other aspects of agency, like action and evaluation of outcomes. The model we propose makes weaker, more realistic assumptions about knowledge, includes a modal operator for belief, and formally represents action and the evaluation of actions as either safe or unsafe.

Recent work at the intersection of game theory and logic focuses on the information flow that occurs during games. Van Ditmarsch identifies a class of games called *knowledge games*, in which players have diverging information [29]. This slightly

relaxes the assumption of classical game theory that players have common knowledge about each other's perfect information. Similarly, it invites logicians to study the information conveyed by the fact that an action is executed. For example, if the action is that agent 1 asks agent 2 the question, "p?", the information conveyed is that 1 does not know whether p, believes that 2 knows whether p, and after the action occurs, this information becomes publicly known. The logic modeling games of this kind is of particular interest to us, as we are concerned with identifying the knowledge and belief state of human pilots based on their actions.

2.3 Logical Framework

Recall that Boolean logic is a simple logic for reasoning about basic propositions using the logical connectives 'and', 'or', 'not', and 'if...then'. It forms the foundation of most logics and has applications ranging from philosophy to circuit design. Propositions are represented as constants p, q and well-formed formulas of the language are constants and any proper combination of constants using the above logical connectives, represented symbolically as,

$$\varphi \stackrel{def}{=} p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \rightarrow \varphi.$$

Boolean logic is limited in what it can express, however. Some situations require higher fidelity, for example when reasoning about knowledge. Consider the proposition, "If Alice knows p, then p is true." Philosophers of knowledge agree with this proposition, considering it a valid claim about knowledge, based on the principle that all known propositions are true. However, this validity cannot be faithfully represented in propositional logic as a general validity. Each sentence in the claim is assigned its own constant, resulting in a formalization like, "$a \rightarrow p$," with a repre-

senting "Alice knows p," and p representing "p is true." But $a \rightarrow p$ is not a validity of Boolean logic.

To represent reasoning about knowledge, and other similar domains requiring higher fidelity, epistemic operators are added to the basic Boolean logic, resulting in the following logic:

$$\varphi \stackrel{def}{=} p \mid \mathbf{K_i}\varphi \mid \neg\varphi \mid \varphi \wedge \varphi,$$

where $\mathbf{K_i}\varphi$ states that agent i *knows* that φ, allowing us to formally represent the above validity:

$$\mathbf{K_{Alice}}p \rightarrow p.$$

Semantics for epistemic logic are given by Kripke structures, which serve as models by which epistemic formulas are evaluated. At its core, a Kripke structure is a graph with nodes and edges, accompanied by a function determining which atomic propositions are true at which worlds. The nodes are normally thought of as possible worlds, or as possible states of the system being modeled. The edges are normally thought of as possibility relations among worlds or states. If, at a node representing a world w, agent A considers it possible that she is in world v, the Kripke semantics modeling this situation would have world w's node connected to world v's node by an edge representing A's epistemic possibility relation.

Formally, we say a Kripke structure is a *tuple* $\langle W,\ V,\ Agents,\ \{R_i | i \in Agents\}\rangle$, where W is a set of worlds, V is a function from propositional constants to sets of worlds satisfying the proposition, $Agents$ is a set of agents, and each R_i is agent i's epistemic possibility relation.

The semantics are as follows, for worlds $w,\ v\ \in W$:

$$w \models p \textbf{ iff } w \in V(p)$$

$$w \models \neg\varphi \textbf{ iff } w \not\models \varphi$$

$$w \models \varphi \wedge \psi \textbf{ iff } w \models \varphi \textit{ and } w \models \psi$$

$$w \models \mathbf{K_i}\varphi \textbf{ iff } \forall v,\ wR_iv \textit{ implies } v \models \varphi.$$

Early applications of epistemic logic in information security modeled system components as agents whose knowledge represented the information flowing to them. Recently, a modified version of epistemic logic known as dynamic epistemic logic has been used in information security to formally reason about security properties involving human components of systems. The research described in this proposal advances along similar lines, treating pilots as human components of safety critical aviation systems and using dynamic epistemic logic to reason about them. The next section describes the basic dynamic epistemic logic.

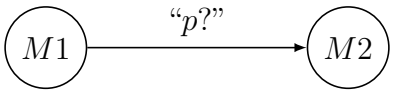
2.3.1 Dynamic Epistemic Logic

Dynamic Epistemic Logic (DEL) formalizes situations in which agents' epistemic states change over time, due to announcements or other informational events. [34] For example, if Alice truthfully and trustworthily communicates to Bob that φ, then after this informational even it is true that Bob knows φ. This situation cannot be modeled by the epistemic logic introduced in the previous section. To model it, we introduce the following formal machinery.

To capture informational events, we introduce the idea of relativizing a Kripke structure. In the previous example, if we model the Alice and Bob situation prior

to Alice's communication, we can have a world w from which Bob considers φ- as well as $\neg\varphi$-worlds possible. However, after the informational event, Bob knows φ, so the model is *relativized* to a submodel in which only φ-worlds are accessible by Bob's epistemic possibility relation. Thus, after the informational event, the model transitions to a submodel with fewer edge relations.

The logic for reasoning about information flow in knowledge games is called Dynamic Epistemic Logic (DEL). As its name suggests, it combines elements of epistemic logic and dynamic logic. Epistemic logic is the static logic for reasoning about knowledge, and dynamic logic is used to reason about actions. In dynamic logic semantics, nodes are states of the system or the world, and relations on nodes are transitions via programs or actions from node to node. If we think of each node in dynamic logic as being a model of epistemic logic, then actions become relations on models, representing transitions from one multi-agent epistemic model to another. For example, if we have a static epistemic model $M1$ representing the knowledge states of agents 1 and 2 at a moment, then the action "p?" is a relation between $M1$ and $M2$, a new static epistemic model of 1's and 2's knowledge after the question is asked. All of this is captured by DEL.



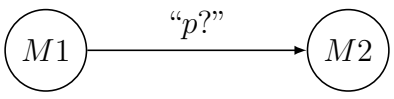
The above figure illustrates the relationship between static epistemic models and dynamic logic models. As a purely dynamic model, the figure shows the action "p?" transitioning between nodes $M1$ and $M2$. If we were to zoom in on the nodes, we would see their structure as epistemic models, with their own nodes and edges, representing possible worlds and epistemic relations.

We are concerned with an additional element: the *safety* status of an action, and an agent's knowledge and belief about that. To capture this, we extend DEL and call the new logic Dynamic Agent Safety Logic (DASL).

Chapter 3

Dynamic Agent Safety Logic

The logic for reasoning about information flow in knowledge games is called Dynamic Epistemic Logic (DEL). As its name suggests, it combines elements of epistemic logic and dynamic logic. Epistemic logic is the static logic for reasoning about knowledge, and dynamic logic is used to reason about actions. In dynamic logic semantics, nodes are states of the system or the world, and relations on nodes are transitions via programs or actions from node to node. If we think of each node in dynamic logic as being a model of epistemic logic, then actions become relations on models, representing transitions from one multi-agent epistemic model to another. For example, if we have a static epistemic model $M1$ representing the knowledge states of agents 1 and 2 at a moment, then the action "p?" is a relation between $M1$ and $M2$, a new static epistemic model of 1's and 2's knowledge after the question is asked. All of this is captured by DEL.



The above figure illustrates the relationship between static epistemic models and dynamic logic models. As a purely dynamic model, the figure shows the action "p?"

transitioning between nodes $M1$ and $M2$. If we were to zoom in on the nodes, we would see their structure as epistemic models, with their own nodes and edges, representing possible worlds and epistemic relations.

We are concerned with an additional element: the *safety* status of an action, and an agent's knowledge and belief about that. To capture this, we extend DEL and call the new logic Dynamic Agent Safety Logic (DASL). The remainder of this section presents DASL's syntax, semantics, and proves its soundness.

3.1 Syntax and Semantics

3.1.1 Syntax

The Dynamic Agent Safety Logic (DASL) used in this paper has the following syntax.

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{K_i}\,\varphi \mid \mathbf{B_i}\,\varphi \mid [\mathbf{i},(\mathbf{A},a)]\varphi \mid [\mathbf{i},(\mathbf{A},a),\mathbf{S}]\varphi,$$

where $p \in AtProp$ is an atomic proposition, $\mathbf{i}$ refers to $i \in Agents$, a is the name of an action, called an action token, belong to a set of such tokens, $Actions$, and $\mathbf{A}$ refers to an action structure. The knowledge operator $\mathbf{K_i}$ indicates that "agent i knows that ..." Similarly, the operator for belief, $\mathbf{B_i}$ can be read, "agent i believes that..." The notion of action tokens and structures will be defined in the semantics. The operators $[\mathbf{i},(\mathbf{A},a)]$ and $[\mathbf{i},(\mathbf{A},a),\mathbf{S}]$ are the dynamic operators for agent i executing action token a from action structure A in the former case, and doing so safely in the latter case. Note that the $\mathbf{S}$ in $[\mathbf{i},(\mathbf{A},a),\mathbf{S}]$ stands for 'safety', and is not a variable, whereas the $\mathbf{i},(\mathbf{A},a)$ are variables for agents, action structures, and action tokens, respectively. One can read the action operators as "after i executes a from A, φ holds.' We define the dual modal operators $\langle\mathbf{K}_i\rangle$, $\langle\mathbf{B}_i\rangle$, $\langle\mathbf{i},(\mathbf{A},a)\rangle$, and $\langle\mathbf{i},(\mathbf{A},a),\mathbf{S}\rangle$ in the usual way.

The semantics of DASL involve two structures that are defined simultaneously, one for epistemic models, and one for action structures capturing the transition relation among epistemic models. Additionally, we define numerous helper functions that straddle the division between metalanguage and object language.

3.1.2 Metalanguage

Kripke Model

A Kripke model $M \in Model$ is a tuple $\langle W, \{R_k^i\}, \{R_b^i\}, w, V\rangle$. It is a set of worlds, sets of epistemic and doxastic relations on worlds for agents, a world denoting the actual world, and a valuation function V mapping atomic propositions to the set of worlds satisfying them. Most readers will be somewhat familiar with epistemic logic, the logic for reasoning about knowledge. Doxastic logic is a similar logic for reasoning about belief[32].

Action Structure

An action structure $A \in ActionStruct$ is a tuple $\langle Actions$,
$\{\chi_k^i\}, \{\chi_b^i\}, a\rangle$. It is a set of action tokens, sets of epistemic and doxastic relations on action tokens for agents, and an action token, a, denoting an actual action token executed.

An action structure captures the associated subjective events of an action occurring, including how it is observed by various agents, incorporating their uncertainty. The action tokens are the actual objective events that might occur. For example, if I am handed a piece of paper telling me who won the Oscar for Best Actress, and I read it, and you see me read it, then the action structure will include possible tokens in which I read that each nominee has won, and you will consider each of these tokens to be possible. When I read the paper, I consider only one action token to be the one

executed. This action structure represents that transition from one epistemic model, in which both of us considers all nominees the potential winner, to an epistemic model in which I know the winner and you still do not know the winner. We can think of the action structure A as the general action "Agent 1 reads the piece of paper" and the tokens as the specific actions "Agent 1 reads that nominee n has won the award."

Model Relation

Just as R_k^i denotes a relation on worlds, $[\![i, (A, a)]\!]$ denotes a relation on Kripke model-world pairs. It represents the relation that holds between M, w and M', w' when agent i executes action (A, a) at M, w and causes the world to transition to M', w'.

Precondition Function

The Precondition function, $pre :: Actions \mapsto \varphi$, maps an action to the formula capturing the conditions under which the action can occur. For example, if we assume agents tell the truth, then an announcement action has as a precondition that the announced proposition is true, as with regular Public Announcement Logic.

Postcondition Function

The Postcondition function, $post :: A \times AtProp \mapsto AtProp$, takes an action structure and an atomic proposition, and maps to the corresponding atomic proposition after the action occurs.

$$post(A, p) = p \text{ if } update(M, A, w, a, i) \models p, \text{ else } \neg p.$$

Update Function

The Update function, $update :: (Model \times ActionStruct \times W \times Actions \times Agents) \mapsto (Model \times W)$, takes a Kripke model M, an action structure A, a world from the Kripke model, an action token from the Action structure, and an agent executing the action, and returns a new Kripke model-world pair. It represents the effect actions have on models, and is more complicated than other DEL semantics in that actions can change the facts on the ground in addition to the knowledge and belief relations. It is a partial function that is defined iff a model-world pair satisfies the action's preconditions.

$update(M, A, w, a, i) = (M', w')$ *where* :

1. $M = \langle W, \{R_k^i\}, \{R_b^i\}, w, V \rangle$
2. $A = \langle Actions, \{\chi_k^i\}, \{\chi_b^i\}, a, pre, post \rangle$
3. $M' = \langle W', \{R_k'^i\}, \{R_b'^i\}, w', V' \rangle$
4. $W' = \{(w, a) | w \in W, a \in Actions, \quad \textsf{and} \quad w \models pre(a)\}$
5. $R_k'^i = \{((w, a), (v, b)) | w R_k^i v \ \textsf{and} \ \ a \chi_k^i b\}$
6. $R_b'^i = \{((w, a), (v, b)) | w R_b^i v \ \textsf{and} \ \ a \chi_b^i b\}$
7. $w' = (w, a)$
8. $V'(p) = post(A, p)$

Safety Precondition Function

The Safety Precondition Function, $pre_s :: Actions \mapsto \varphi$, is a more restrictive function than pre. Where pre returns the conditions that dictate whether the action is possible, pre_s returns the conditions that dictate whether the action is safely permissible. This function is the key reason the dynamic approach allows for easy inference from action to safety-critical information.

3.1.3 Semantics

The logic DASL has the following Kripke semantics.

$$
\begin{aligned}
M, w \models p \text{ iff } & \ w \in V(p) \\
M, w \models \neg\varphi \text{ iff } & \ M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi \text{ iff } & \ M, w \models \varphi \text{ and } \ M, w \models \psi \\
M, w \models \mathbf{K_i}\,\varphi \text{ iff } & \ \forall v,\ wR^i_k v \ \text{ implies } \ M, v \models \varphi \\
M, w \models \mathbf{B_i}\,\varphi \text{ iff } & \ \forall v,\ wR^i_b v \ \text{ implies } \ M, v \models \varphi \\
M, w \models [\mathbf{i}, (\mathbf{A}, a)]\varphi \text{ iff } & \ \forall M', w',\ (M, w)[\![i, (A, a)]\!](M', w') \\
& \text{implies } \ M', w' \models \varphi \\
M, w \models [\mathbf{i}, (\mathbf{A}, a), \mathbf{S}]\varphi \text{ iff } & \ \forall M', w',\ (M, w)[\![i, (A, a), S]\!](M', w') \\
& \text{implies } \ M', w' \models \varphi
\end{aligned}
$$

The definitions of the dynamic modalities make use of a relation between two model-world pairs, which we now define.

$$
\begin{aligned}
(M, w)[\![i, (A, a)]\!](M', w') \text{ iff } & \ M, w \models pre(a) \\
& \text{and } \ update(M, A, w, a, i) = (M', w') \\
(M, w)[\![i, (A, a), S]\!](M', w') \text{ iff } & \ M, w \models pre_s(a) \\
& \text{and } \ update(M, A, w, a, i) = (M', w')
\end{aligned}
$$

3.1.4 Hilbert System

DASL is axiomatized by the following Hilbert system.

All propositional tautologies are axioms.

$\mathbf{K_i}$ is T (knowledge relation is reflexive)

$\mathbf{B_i}$ is KD45 (belief relation is serial, transitive, and Euclidean)

EP1: $\mathbf{K_i}\,\varphi \Rightarrow \mathbf{B_i}\,\varphi$

EP2: $\mathbf{B_i}\,\varphi \Rightarrow \mathbf{B_i}\,\mathbf{K_i}\,\varphi$

EP3: $\mathbf{B_i}\,\varphi \Rightarrow \mathbf{K_i}\,\mathbf{B_i}\,\varphi$

SP: $[\mathbf{i}, (\mathbf{A}, a)]\varphi \Rightarrow [\mathbf{i}, (\mathbf{A}, a), \mathbf{S}]\varphi$

PR: $\langle \mathbf{i}, (\mathbf{A}, a)\rangle\,\varphi \Rightarrow \mathbf{B_i}\,\langle \mathbf{i}, (\mathbf{A}, a), \mathbf{S}\rangle\,\varphi$,

plus the inference rules Modus Ponens and Necessitation for $\mathbf{K_i}$ and $\mathbf{B_i}$.

Above are the axioms characterizing the logic. Knowledge is weaker here than in most epistemic logics, and belief is standard [30]. They are related logically by EP(1-3), which hold that knowledge entails belief, belief entails that one believes that one knows, and belief entails than one knows that one believes. Finally, actions and safe actions are logically related by SP and PR, which hold that necessary consequences of *mere* action are also necessary consequences of *safe* actions, and that a pilot can execute an action only if he believes that he is executing a safe action.

Below are the axioms characterizing the reduction laws from the dynamic logic to a purely static logic through recursive application.

Aprop: $[\mathbf{i},(\mathbf{A},a)]p \Leftrightarrow (pre(a) \Rightarrow (post(A,p) \Rightarrow p))$

AN: $[\mathbf{i},(\mathbf{A},a)]\neg\varphi \Leftrightarrow (pre(a) \Rightarrow \neg[\mathbf{i}(\mathbf{A},a)]\varphi)$

AC: $[\mathbf{i},(\mathbf{A},a)](\varphi \wedge \psi) \Leftrightarrow ([\mathbf{i},(\mathbf{A},a)]\varphi \wedge [\mathbf{i},(\mathbf{A},a)]\psi)$

AK: $[\mathbf{i},(\mathbf{A},a)]\mathbf{K_i}\,\varphi \Leftrightarrow (pre(a) \Rightarrow \bigwedge_{a\chi^i_k b} \mathbf{K_i}\,[\mathbf{i},(\mathbf{A},b)]\varphi)$

AB: $[\mathbf{i},(\mathbf{A},a)]\mathbf{B_i}\,\varphi \Leftrightarrow (pre(a) \Rightarrow \bigwedge_{a\chi^i_b b} \mathbf{B_i}\,[\mathbf{i},(\mathbf{A},b)]\varphi)$

Sprop: $[\mathbf{i},(\mathbf{A},a),\mathbf{S}]p \Leftrightarrow (pre_s(a) \Rightarrow (post(A,p) \Rightarrow p))$

SN: $[\mathbf{i},(\mathbf{A},a),\mathbf{s}]\neg\varphi \Leftrightarrow (pre_s(a) \Rightarrow \neg[\mathbf{i}(\mathbf{A},a),\mathbf{S}]\varphi)$

SC: $[\mathbf{i},(\mathbf{A},a),\mathbf{S}](\varphi \wedge \psi) \Leftrightarrow ([\mathbf{i},(\mathbf{A},a),\mathbf{S}]\varphi \wedge [\mathbf{i},(\mathbf{A},a),\mathbf{S}]\psi)$

SK: $[\mathbf{i},(\mathbf{A},a),\mathbf{S}]\mathbf{K_i}\,\varphi \Leftrightarrow (pre_s(a) \Rightarrow \bigwedge_{aR'^i_k b} \mathbf{K_i}\,[\mathbf{i},(\mathbf{A},b),\mathbf{S}]\varphi)$

SB: $[\mathbf{i},(\mathbf{A},a),\mathbf{S}]\mathbf{B_i}\,\varphi \Leftrightarrow (pre_s(a) \Rightarrow \bigwedge_{aR'^i_b b} \mathbf{B_i}\,[\mathbf{i},(\mathbf{A},b),\mathbf{S}]\varphi)$

3.2 Soundness

Theorem 3.2.1 (Soundness)**.** *Dynamic Agent Safety Logic is sound for Kripke structures with*

(1) reflexive R^i_k relations,

(2) serial, transitive, Euclidean R^i_b relations,

(3) which are partially ordered $(R^i_k \circ R^i_b) \subseteq R^i_b$, $(R^i_b \circ R^i_k) \subseteq R^i_b$, and $R^i_b \subseteq R^i_k$,

(4) $[\![i,(A,a),S]\!] \subseteq [\![i,(A,a)]\!]$and

(5) $([\![i,(A,a),S]\!] \circ R^i_b) \subseteq [\![i,(A,a)]\!]$.

Proof. (1) *and* (2) correspond to the axioms that $\mathbf{K_i}$ is a T modality and $\mathbf{B_i}$ is a KD45 modality in the usual way. (3) corresponds to EP1, EP2, and EP3. Axioms AP through SB are reduction axioms. This leaves (4), corresponding to SP, and (5) which corresponds to PR. Here we will prove (5). Let M be a Kripke structure satisfying the five conditions above. Let A be an Action structure with a and i as its actual

action token and agent.

We prove (5) via the contrapositive of PR: $\langle \mathbf{B}_i \rangle [\mathbf{i}, (\mathbf{A}, a), \mathbf{S}]\varphi \Rightarrow [\mathbf{i}, (\mathbf{A}, a)]\varphi$. Assume $M, w \models \langle \mathbf{B}_i \rangle [\mathbf{i}.(\mathbf{A}, a), \mathbf{S}]\varphi$. By the semantics of $\langle \mathbf{B}_i \rangle$, there exists a v, such that $wR_b^i v$ and $v \models [\mathbf{i}, (\mathbf{A}, a), \mathbf{S}]\varphi$. From the semantics, it follows that forall M', v', if $(M, v)[\![i, (A, a), S]\!](M', v')$ then $M', v' \models \varphi$. By slightly abusing the notation, and letting $(W, w)R_b^i(W, v)$ be equivalent to $wR_b^i v$, we can create the composed relation $([\![i, (A, a), S]\!] \circ R_b^i)$. It then holds, by condition (5), that $(M, w)([\![i, (A, a), S]\!] \circ R_b^i)(M', v')$ implies $(M, w)[\![i, (A, a)]\!](M', v')$. So, for all M', v', if $(M, w)[\![i, (A, a)]\!](M', v')$, then $M', v' \models \varphi$. So, $M, w \models [\mathbf{i}, (\mathbf{A}, a)]\varphi$. $\square$

Aprop : $\Rightarrow$. Assume $M, w \models [\mathbf{i}, (\mathbf{A}, a)]p$. We must show that $M, w \models pre(a) \Rightarrow (post(A, p) \Rightarrow$ By the semantics of $[\mathbf{i}, (\mathbf{A}, a)]$, for all (M', w'), if $M, w \models pre(a)$ and $update(M, A, w, a, i) = (M', w')$, then $M', w' \models p$. By definition of $post(A, p)$, if $update(M, A, w, a, i) = (M', w')$ and $M', w' \models p$, then $post(A, p) = p$. So, if $M, w \models pre(a)$, then $post(A, p) = p$, and thus $post(A, p) \Rightarrow p$.

$\Leftarrow$. Assume $M, w \models pre(a) \Rightarrow (post(A, p) \Rightarrow p)$. By the definition of $post(A, p)$, if $post(A, p) = p$ then $update(M, A, w, a, i) \models p$. So, if $M, w \models pre(a)$, then $update(M, A, w, a, i) \models p$. Therefore, $M, w \models [\mathbf{i}, (\mathbf{A}, a)]p$.

AN : $\Rightarrow$. Assume $M, w \models [\mathbf{i}, (\mathbf{A}, a)]\neg\varphi$. It suffices to show that $M, w \models pre(a) \Rightarrow \langle \mathbf{i}, (\mathbf{A}, a) \rangle \neg\varphi$. From the assumption and the semantics, for all (W', w'), if $(M, w)[\![i, (A, a)]\!](M', w')$ then $M', w' \models \neg\varphi$. So, if $M, w \models pre(a)$ and $update($ $M, A, w, a, i) = (M', w')$, then $M', w' \models \neg\varphi$. Assume $M, w \models pre(a)$, and it follows that $update(M, A, w, a, i)$ is defined, so there exists a M', w' such that $(M, w)[\![i, (A, a)]\!](M', w')$ and $update(M, A, i) = (M', w')$ and $M', w' \models \neg\varphi$. Therefore, $M, w \models pre(a) \Rightarrow \langle \mathbf{i}, (\mathbf{A}, a) \rangle \neg\varphi$.

$\Leftarrow$. Assume $M, w \models pre(a) \Rightarrow \neg[\mathbf{i}, (\mathbf{A}, a)]\varphi$. This is equivalent to $M, w \models pre(a) \Rightarrow \langle \mathbf{i}, (\mathbf{A}, a) \rangle \neg\varphi$. By the semantics, if $M, w \models pre(a)$, then there exists a (M', w') such that $(M, w)[\![i, (A, a)]\!](M', w')$ and $M', w' \models \neg\varphi$. The relation $[\![i, (A, a)]\!]$ is functional, so $\exists$ implies $\forall$. So, for all (M', w'), if $(M, w)[\![i, (A, a)]\!](M',$

w'), then $M', w' \models \neg\varphi$, and therefore $M, w \models [\mathbf{i}, (\mathbf{A}, a)]\neg\varphi$.

AC is obvious.

AK. For this proof, assume for simplicity, without loss of generality, that $Actions = \{a\}$.

$\Rightarrow$. Assume $M, w \models [\mathbf{i}, (\mathbf{A}, a)]\mathbf{K_i}\,\varphi$. Unfolding the semantics, for all (M', w'), if $(M, w) \models pre(a)$ and $update(M, A, w, a,$
$i) = (M', v')$, then $M', w' \models \mathbf{K_i}\,\varphi$. $M', w' \models \mathbf{K_i}\,\varphi$ iff for all $v \in W$, if $wR^i_k v$ and $M, v \models pre(a)$ and $update(M, A, v, a, i) = (M', v')$ and $a\chi^i_k a$, then $M', v' \models \varphi$. That is, $M, w \models \mathbf{K_i}\,[\mathbf{i}, (\mathbf{A}, a)]\varphi$.

$\Leftarrow$. Assume $M, w \models pre(a) \Rightarrow \mathbf{K_i}\,[\mathbf{i}, (\mathbf{A}, a)]\varphi$. We must show $M, w \models [\mathbf{i}, (\mathbf{A}, a)]$ $\mathbf{K_i}\,\varphi$. Thus, we must show $M, w \models pre(a)$ and $update(M, A, w, a, i) = (M', w')$ implies $M', w' \models \mathbf{K_i}\,\varphi$. So it suffices to show that if $update(M, A, w, a, i) = (M', w')$ and $M, w \models \mathbf{K_i}\,[\mathbf{i}, (\mathbf{A}, a)]\varphi$, then $M', w' \models \mathbf{K_i}\,\varphi$. Assume $update(M, A, w,$
$a, i) = (M', w')$ and $M, w \models \mathbf{K_i}\,[\mathbf{i}, (\mathbf{A}, a)]\varphi$. Then for all v, if $wR^i_k v$, then $M, v \models [\mathbf{i}, (\mathbf{A}, a)]\varphi$. It follows that $M, v \models pre(a)$ and $update(M, A, v, a, i) = (M', v')$ implies $M', v' \models \varphi$. Since $wR^i_k v$ and $a\chi^i_k a$, it holds that $w'R^{i'}_k v'$. Thus, $M', w' \models \mathbf{K_i}\,\varphi$.

Proofs for **AB** through **SB** follow the above proofs exactly analogously. □

Assume $M, w \models \langle \mathbf{i}, a\rangle\, true$. By the semantics of $\langle \mathbf{i}, a\rangle$, $M, w \models pre(a)$ and $update(M, w, \chi) \models true$. Let $(M', w') = update(M, w, \chi)$. Then $M', w' \models true$. From (5) above, it holds that $R^i_b(w) \subseteq V^s(a)$. R_b is serial, so there is at least one such $v \in R^i_b$. Then $M, v \models pre_s(a)$. From (4), $M, v \models pre(a)$, so $update(M, v, \chi)$ is defined, call it (M'', v'). Because (M'', v') is defined, $M'', v' \models true$. So, $M, v \models pre_s(a)$ and $update(M, v, \chi) \models true$. This holds for all v, such that $wR_b v$. Thus, $M, w \models \mathbf{B_i}\,\langle \mathbf{i}, a, \mathbf{S}\rangle\, true$. Therefore, $M, w \models \langle \mathbf{i}, a\rangle\, true \Rightarrow \mathbf{B_i}\,\langle \mathbf{i}, a, \mathbf{S}\rangle\, true$. □

Next we turn to completeness.

3.2.1 Completeness

Theorem 3.2.2 (Completeness)**.** *The language of Dynamic Agent Safety Logic, $\mathcal{L}_{DASL}$, is complete for Kripke structures with*

(1) reflexive R_k^i relations,

(2) serial, transitive, Euclidean R_b^i relations,

(3) which are partially ordered $(R_k^i \circ R_b^i) \subseteq R_b^i$, $(R_b^i \circ R_k^i) \subseteq R_b^i$, and $R_b^i \subseteq R_k^i$,

(4) $[\![i, (A, a), S]\!] \subseteq [\![i, (A, a)]\!]$ and

(5) $([\![i, (A, a), S]\!] \circ R_b^i) \subseteq [\![i, (A, a)]\!]$.

Proof Sketch. Completeness states that if a formula φ is valid, then it is deducible. The sketch for this proceeds proceeds by contraposition.

Proof Goal. For every formula φ in the language $\mathcal{L}_{DASL}$, $\nvdash \varphi$ implies $\not\models \varphi$.

We define the notions of a *Maximal Consistent Set*, and a *Canonical Model* for the logic.

Definition. Maximal Consistent Set.

For a set of formulas $\Gamma \subseteq \mathcal{L}_{DASL}$, Γ is maximal consistent iff

1. Γ is consistent: $\Gamma \nvdash \bot$.

2. Γ is maximal: there is no $\Gamma' \subseteq \mathcal{L}_{DASL}$ such that $\Gamma \subset \Gamma'$ and $\Gamma' \vdash \bot$.

Definition. Canonical Model. A canonical model $M^C = \langle W^C,\ R_{k,i}^C,\ R_{b,i}^C,\ w,\ V^C \rangle$ is defined:

1. $W^C = \{\Gamma | \Gamma \text{ is maximal consistent}\}$

2. $\Gamma R_{k,i}^C \Delta$ iff $\{\mathbf{K_i}\,\varphi | \mathbf{K_i}\,\varphi \in \Gamma\} = \{\mathbf{K_i}\,\varphi | \mathbf{K_i}\,\varphi \in \Delta\}$

3. $\Gamma R_{b,i}^C \Delta$ iff $\{\mathbf{B_i}\,\varphi | \mathbf{B_i}\,\varphi \in \Gamma\} = \{\mathbf{B_i}\,\varphi | \mathbf{B_i}\,\varphi \in \Delta\}$

4. $V^C = \{\Gamma \in W^C | \Gamma \subseteq V(p)\}$

The proof appeals to the following lemmas, proven in [34]:

Lindenbaum: Every consistent set of formulas is a subset of a maximal consistent set of formulas.

Properties: If Γ and Δ are maximal consistent sets, then:

- 1. Γ and Δ are deductively closed.
- 2. $\varphi \in \Gamma$ iff $\neg\varphi \notin \Gamma$.
- 3. $(\varphi \wedge \psi) \in \Gamma$ iff $\varphi \in \Gamma$ and $\psi \in Gamma$.
- 4. $\Gamma R^C_{k,i} \Delta$ iff $\{\mathbf{K_i}\,\varphi | \mathbf{K_i}\,\varphi \in \Gamma\} \subseteq \Delta$.
- 5. $\Gamma R^C_{b,i} \Delta$ iff $\{\mathbf{B_i}\,\varphi | \mathbf{B_i}\,\varphi \in \Gamma\} \subseteq \Delta$.
- 6. $\{\mathbf{K_i}\,\varphi | \mathbf{K_i}\,\varphi \in \Gamma\} \vdash \psi$ iff $\{\mathbf{K_i}\,\varphi | \mathbf{K_i}\,\varphi \in \Gamma\} \vdash \mathbf{K_i}\,\psi$.

Truth: For every $\varphi \in \mathcal{L}_{DASL}$, and every maximal consistent set Γ:

$$\varphi \in \Gamma \text{ iff } (M^C, \Gamma) \models \varphi$$

Canonicity: The canonical model satisfies the above frame conditions.
With these lemmas, we assume $\nvdash \varphi$, and show that $\not\models \varphi$. Since $\nvdash \varphi$, the set $\{\neg\varphi\}$ is consistent. From Lindenbaum, $\{\neg\varphi\}$ is part of a maximal consistent set, call it Γ. From the Truth Lemma, $(M^C, \Gamma) \models \neg\varphi$. Therefore, $\not\models \varphi$.$\Box$

Because the static logic is complete and we have translation axioms that convert the dynamic formulas to equivalent static ones, we can conclude that the entirety of DASL is complete. Next we examine case studies and a mechanization of the logic.

Chapter 4

Case Study and Mechanization

We apply the logic just developed to the formal analysis of the Air France 447 aviation incident. We also mechanize the formalization in the Coq Proof Assistant. Our mechanization follows similar work by Maliković and Čubrilo [39, 40], in which they mechanize an analysis of the game of Cluedo using Dynamic Epistemic Logic, based on van Ditmarsch's formalization of the game [29]. It is commonly assumed that games must be adversarial, but this is not the case. Games need only involve situations in which players' payoffs depend on the actions of other players. Similarly, knowledge games need not be adversarial, and must only involve diverging information. Thus, it is appropriate to model aviation incidents as knowledge games of sorts, where players' payoffs depend on what others do, specifically the way the players communicate information with each other. The goal is to achieve an accurate situational awareness and provide flight control inputs appropriate for the situation. Failures to achieve this goal result in disaster, and often result from imperfect information flow. A formal model of information flow in these situations provides insight and allows for the application of formal methods to improve information flow during emergency situations.

4.1 Air France 447

This case study is based on the authoritative investigative report into Air France 447 performed and released by France's Bureau d'Enquêtes et d'Analyses pour la Sécurité de l'Aviation Civile (BEA), responsible for investigating civil aviation incidents and issuing factual findings[19]. The case is mechanized by instantiating, in Coq, the above logic to reflect the facts of the case. One challenge associated with this is that the readings about inputs present in aviation are often real values on a continuum, whereas for our purposes we require discrete values. We accomplish this by dividing the continuum associated with inputs and readings into discrete chunks, similar to how fuzzy logic maps defines predicates with real values[37].

Air France flight 447 from Rio de Janeiro, Brazil to Paris, France, departed June 1, 2009. The Airbus A330 encountered adverse weather over the Atlantic ocean, resulting in a clogged Pitot-static system. Consequently, the airspeed indicators delivered unreliable data concerning airspeed to the pilot flying, resulting in confusion. A chain of events transpired in which the pilot overcorrected the plane's horizontal attitude again and again, and continued to input nose up pitch commands, all while losing airspeed. Perhaps most confusing to the pilot was the following situation: the aircraft's angle of attack (AOA) was so high it was considered invalid by the computer, so no stall warning sounded until the nose pitched down into the valid AOA range, at which point the stall warning would sound. When the pilot pulled up, the AOA would be considered invalid again, and the stall warning would cease. The aircraft entered a spin and crashed into the ocean. Palmer [42] argues that had the pilot merely taken no action, the Pitot tubes would have cleared in a matter of seconds, and the autopilot could have returned to Normal Law.

This section will formalize an excerpted instance from the beginning of the case, involving an initial inconsistency among airspeed indicators, and the subsequent dangerous input provided by the pilot. Formalized in the logic, the facts of the case allow

us to infer that the pilot lacked negative introspection about the safety-critical data required for his action. This demonstrates that the logic allows information about the pilot's situational awareness to flow to the computer, via the pilot's actions. It likewise establishes a safety property to be enforced by the computer, namely that a pilot should maintain negative introspection about safety-critical data, and if he fails to do so, it should be re-established as quickly as possible.

1. $\neg(RS = LS) \wedge \neg(mode = normal) \ldots$
2. $\langle pilot, hardnoseup \rangle \, true$
3. $\mathbf{B}_{pilot} \, (LS = RS)$
4. $\neg \mathbf{K}_{pilot} \, (LS = RS)$
5. $\mathbf{B}_{pilot} \, \mathbf{K}_{pilot} \, (LS = RS)$
6. $\neg \mathbf{K}_{pilot} \, \neg \mathbf{K}_{pilot} \, (LS = RS)$
7. $\neg \mathbf{K}_{pilot} \, (LS = RS) \wedge \neg \mathbf{K}_{pilot} \, \neg \mathbf{K}_{pilot} \, (LS = RS)$
8. $\neg(\neg \mathbf{K}_{pilot} \, (LS = RS) \Rightarrow \mathbf{K}_{pilot} \, \neg \mathbf{K}_{pilot} \, (LS = RS))$

Premise 1 is a conjunction of the current instrument readings, wherein the right side airspeed indicator and the left side airspeed indicator do not indicate the same speed, and the mode is not normal. Premise 2 holds when the pilot executes the action giving the input of a hard nose up pitch. 3 follows from 2 by axiom *SP* and the semantics of the safe action modality. 4 follows from 1 and the fact that the knowledge operator is reflexive. 5 follows from 3 and axiom *EP2*. 6 follows from the fact that the belief modality is serial, and the contrapositive of axiom *EP1*. 7 follows from 6 and 4. 8 follows from 7, as they are logically equivalent. The above argument shows that from the configuration of the instruments and the pilot's action,

it is deducible that the pilot lacks negative introspection about the airspeed indicator readings.

The above formalization of the case focuses on one action, and deduces one of the pieces of safety-critical information. A similar deduction follows for the pilot's unawareness of $(\neg(mode = normal))$. Similar modeling can be done involving pilot announcements to each other. Furthermore, should the autopilot be modeled, and if it has corrective actions available to it, these and their effects can be modeled as well. The next section mechanizes this model in the Coq Proof Assistant.

4.2 Mechanization in Coq

The following mechanization demonstrates progress from the artificially simple toy examples normally analyzed in the literature to richer real-world examples. However, it does not represent the full richness of the approach. The actions and instrument readings mechanized in this paper are constrained to those most relevant to the case study. The approach is capable of capturing the full richness of all instrument reading configurations and actions available to a pilot. To do so, one needs to consult a flight safety manual and formally represent each action available to a pilot, and each potential instrument reading, according to the following scheme.

Before beginning, we note that our use of sets in the following Coq code requires the following argument passed to coqtop before executing: -impredicative-set. In CoqIDE, this can be done by selecting the 'Tools' dropdown, then 'Coqtop arguments'. Type in *-impredicative-set*.

We first formalize the set of agents.

```
Inductive Agents: Set := Pilot | CoPilot | AutoPilot.
```

Next we formalize the set of available inputs. These themselves are not actions, but represent atomic propositions true or false of a configuration.

```
Inductive Inputs : Set :=
HardThrustPlus | ThrustPlus
| HardNoseUp    | NoseUp
| HardWingLeft  | WingLeft
| HardThrustMinus | ThrustMinus
| HardNoseDown  | NoseDown
| HardWingRight | WingRight.
```

We represent readings by indicating which *side* of the panel they are on. Typically, an instrument has a left-side version, a right-side version, and sometimes a middle version serving as backup. When one of these instruments conflicts with its siblings, the autopilot will disconnect and give control to the pilot.

```
Inductive Side : Set := Left | Middle | Right.
```

We divide the main instruments into chunks of values they can take, in order to provide them with a discrete representation in the logic. For example, the reading *VertUp1* may represent a nose up reading between 0° and 10°, while *VertUp2* represents a reading between 11° and 20°.

```
Inductive Readings (s : Side) : Set :=
VertUp1 | VertUp2 | VertUp3 | VertUp4
| VertDown1 | VertDown2 | VertDown3 | VertDown4
| VertLevel | HorLeft1 | HorLeft2 | HorLeft3
| HorRight1 | HorRight2 | HorRight3 | HorLevel
| AirspeedFast1 | AirspeedFast2 | AirspeedFast3
| AirspeedSlow1 | AirspeedSlow2 | AirspeedSlow3
| AirspeedCruise| AltCruise | AltClimb | AltDesc | AltLand.
```

We define a set of potential modes the aircraft can be in.

```
Inductive Mode : Set := Normal | Alternate1 | Alternate2.
```

We define a set of global instrument readings representing the mode and all of the instrument readings, left, right, and middle, combined together. This represents the configuration of the instumentation.

```
Inductive GlobalReadings : Set := Global (m: Mode)
(rl : Readings Left)
(rm : Readings Middle)
(rr : Readings Right).
```

The set of atomic propositions we are concerned with are those representing facts about the instrumentation.

```
Inductive Atoms : Set :=
| M (m : Mode)
| Input (a : Inputs)
| InstrumentL (r : Readings Left)
| InstrumentM (r : Readings Middle)
| InstrumentR (r : Readings Right)
| InstrumentsG (g : GlobalReadings).
```

Next we follow Maliković and Čubrilo [39, 40] in defining a set *prop* of propositions in predicate calculus, distinct from Coq's built in type *Prop*. The definition provides constructors for atomic propositions consisting of particular instrument reading predicate statements, implications, propositions beginning with a knowledge modality, and those beginning with a belief modality. Interestingly, modal logic cannot be directly represented in Coq's framework [38]. We first define propositions in first-order logic, which we then use to define DASL. This appears to be the standard technique for mechanizing modal logics in Coq.

```
Inductive prop : Set :=
| atm : Atoms → prop
| imp: prop → prop → prop
| Forall : forall (A : Set), (A → prop) → prop
| K : Agents → prop → prop
| B : Agents → prop → prop
| Ck : list Agents → prop → prop
| Cb : list Agents → prop → prop.
```

We use the following notation for implication and universal quantification.

```
Infix "⟹ " := imp (right associativity, at level 85).
Notation "\−/ p" := (Forall _ p) (at level 70, right associativity).
```

We likewise follow Maliković and Čubrilo [39, 40] by defining an inductive type *theorem* representing a theorem of DASL. The constructors correspond to the Hilbert system, either as characteristic axioms, or inference rules. The first three represent axioms for propositional logic, then the rule Modus Ponens, then the axioms for the epistemic operator plus its Necessitation rule, then the doxastic operator and its Necessitation rule. Do not confuse the Necessitation rules with material implication in the object language. The final constructors capture the axioms relating belief and knowledge. The axioms for dynamic modal operators are defined separately, and are not included here.

```
Inductive theorem : prop → Prop :=
|Hilbert_K: forall p q : prop, theorem (p ⟹ q ⟹ p)
|Hilbert_S: forall p q r : prop,
theorem ((p⟹ q⟹ r)⟹ (p⟹ q)⟹ (p⟹ r))
|Classic_NOTNOT : forall p : prop, theorem ((NOT (NOT p)) ⟹ p)
|MP : forall p q : prop, theorem (p ⟹ q) → theorem p → theorem q
|K_Nec : forall (a : Agents) (p : prop), theorem p → theorem (K a p)
|K_K : forall (a : Agents) (p q : prop),
theorem (K a p ⟹ K a (p ⟹ q) ⟹ K a q)
|K_T : forall (a : Agents) (p : prop), theorem (K a p ⟹ p)
|B_Nec : forall (a : Agents) (p : prop), theorem p → theorem (B a p)
|B_K : forall (a : Agents) (p q : prop),
theorem (B a p ⟹ B a (p ⟹ q) ⟹ B a q)
|B_Serial : forall (a : Agents) (p : prop),
theorem (B a p ⟹ NOT (B a (NOT p)))
|B_4 : forall (a : Agents) (p : prop), theorem (B a p ⟹ B a (B a p))
|B_5 : forall (a : Agents) (p : prop),
theorem (NOT (B a p) ⟹ B a (NOT (B a p)))
|K_B : forall (a : Agents) (p : prop), theorem (K a p ⟹ B a p)
|B_BK : forall (a : Agents) (p : prop), theorem (B a p ⟹ B a (K a p)).
```

We use the following notation for *theorem*:

```
Notation "|-- p" := (theorem p) (at level 80).
```

We encode actions as records in Coq, recording the acting pilot, the observability of the action (whether it is observed by other agents or not), the input provided by the pilot, and the preconditions for the action and the safety preconditions for the action, both represented as global atoms.

```
Record Action : Set := act {Ai : Agents; Aj : Agents; pi : PI;
input : Inputs; c : GlobalReadings;
c_s : GlobalReadings}.
```

The variable c holds the configuration representing the precondition for the action, while the variable c_s holds the configuration for the safety precondition.

We encode the precondition and safety precondition functions as follows.

```
Function pre (a:Action) : prop := atm (InstrumentsG (c a)).
Function pre_s (a : Action) : prop := atm (InstrumentsG (c_s a)).
```

In the object language, the dynamic modalities of action and safe action are encoded as follows.

```
Parameter aft_ex_act : Action → prop → prop.
Parameter aft_ex_act_s : Action → prop → prop.
```

Many standard properties of logic, like the simplification of conjunctions, hypothetical syllogism, and contraposition, are encoded as Coq axioms. As an example, here is how we encode simplifying a conjunction into just its left conjunct.

```
Axiom simplifyL : forall p1 p2,
|-- p1 & p2 → |-- p1.
```

We formalize the configuration of the instruments at 2 hour 10 minutes into the flight as follows.

```
Definition Config_1 := (atm (M Alternate2)) &
(atm (InstrumentL (AirspeedSlow3 Left))) &
(atm (InstrumentM (AirspeedSlow3 Middle))) &
(atm (InstrumentR (AirspeedCruise Right))).
```

The mode is Alternate Law 2, and the left and central backup instruments falsely indicate that the airspeed is very slow, while the right side was not recorded, but because there was a conflict, we assume it remained correctly indicating a cruising airspeed.

The pilot's dangerous input, a hard nose up command, is encoded as follows.

```
Definition Input1 := act Pilot Pilot Pri HardNoseUp
(Global Alternate2 (AirspeedSlow3 Left)
(AirspeedSlow3 Middle)
(AirspeedCruise Right))
(Global Normal (AirspeedCruise Left)
(AirspeedCruise Middle)
(AirspeedCruise Right)).
```

The action is represented in the object language by taking the dual of the dynamic modality, $\neg[\mathbf{i}, (\mathbf{A}, a)]\neg True$, equivalently $\langle \mathbf{i}, (\mathbf{A}, a)\rangle True$, indicating that the precondition is satisfied and the action token is executed.

```
Definition Act_1 := NOT (aft_ex_act Input1 (NOT TRUE)).
```

The actual configuration satisfies the precondition for the action, but it is inconsistent with the safety precondition. The safety precondition for the action indicates that the mode should be Normal and the readings should consistently indicate cruising airspeed. However, in Config_1, the conditions do not hold. Thus, the action is unsafe. From the configuration and the action, DASL allows us to deduce that the pilot lacks negative introspection of the action's safety preconditions.

Negative introspection is an agent's awareness of the current unknowns. To lack it is to be unaware of one's unknown variables, so lacking negative introspection about one's safety preconditions is to be unaware that they are unknown.

```
Theorem NegIntroFailMode :
|-- (Config_1 ⟹ Act_1 ⟹
((NOT (K Pilot (pre_s(Action1)))) &
(NOT (K Pilot (NOT (K Pilot (pre_s(Action1))))))))).
```

In fact, in general it holds that if the safety preconditions for an action are false, and the pilot executes that action, then the pilot lacks negative introspection of those conditions. We have proven both the above theorem, and the more general theorem, in Coq.

```
Theorem neg_intro_failure :
forall (A Ao : Agents) (pi : PI) (inp : Inputs)
(m : Mode)
(rl : Readings Left) (rm : Readings Middle) (rr : Readings Right)
(ms : Mode)
(rls : Readings Left) (rms : Readings Middle) (rrs : Readings Right)
phi,
|-- (NOT
(aft_ex_act
(act A Ao pi inp (Global m rl rm rr) (Global ms rls rms rrs))
(NOT phi)) ⟹
NOT (atm (InstrumentsG (Global ms rls rms rrs))) ⟹
(NOT (K A (atm (InstrumentsG (Global ms rls rms rrs)))) &
(NOT (K A (NOT (K A (atm (InstrumentsG (Global ms rls rms rrs)))))))).
```

This indicates that negative introspection about safety preconditions is a desirable safety property to maintain, consistent with the official report's criticism that the Airbus cockpit system did not clearly display the safety critical information. The logic described in this research accurately models the report's findings that the pilot's lack of awareness about safety-critical information played a key role in his decision to provide what turned out to be unsafe inputs. Furthermore, the logic supports efforts to automatically infer which safety-critical information the pilot is unaware of and effectively display it to him.

4.3 Advantages of this Approach

Compared to earlier efforts to bring formal methods to bear on pilot error, this approach has several differences and advantages.

1. The formalization of the pilot's mental model is generated by her own behavior
2. It better copes with the unpredictability of human behavior
3. It is system non-specific

As mentioned in Section **??**, earlier efforts to bring formal methods to bear do not directly model the human's mental model. This approach complements those by doing so. Similarly, rather than generating a behavior model of the user at specification-time and seeking to prevent bad runs, this approach works in the opposite direction. By formalizing the analyses produced by decades of aviation mishap analysis, safety properties can be identified. Just as the accident investigators read through data from prior to the crash to figure out what went wrong, a computational approach equipped with DASL can draw the same inferences as the investigators, but immediately after the bad action occurs, and hopefully before the plane crashes. Thus, the myriad ways a human can mess up need not all be modeled at specification time, but merely expressible at runtime, as this approach allows. Because DASL is not designed with one specific aircraft in mind, its safety preconditions can be tailored to any system, beyond even aviation. Any system with a human component, rigorously definable guidelines for proper action, and a computer with a global view of the system, can be modeled by DASL.

Chapter 5

Application: Runtime Monitor

5.1 Formal Analysis of Pilot Error

In [**?**], the author develops a Dynamic Agent Safety Logic (DASL) for analyzing safety-critical information flow during aviation mishaps. It is a modal logic based on Dynamic Epistemic Logic (DEL), a dynamic modal logic for reasoning about how knowledge and actions are related [34]. The Dynamic Agent Safety Logic (DASL) used in this paper has the following syntax.

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{K_i}\,\varphi \mid \mathbf{B_i}\,\varphi \mid [\mathbf{i}, a]\varphi \mid [\mathbf{i}, a, \mathbf{S}]\varphi,$$

where $p \in AtProp$ is an atomic proposition, $\mathbf{i}$ refers to $i \in Agents$, and $a \in Actions$. The knowledge operator $\mathbf{K_i}$ indicates that "agent i knows that ..." Similarly, the operator for belief, $\mathbf{B_i}$ can be read, "agent i believes that..." The operators $[\mathbf{i}, a]$ and $[\mathbf{i}, a, \mathbf{S}]$ are the dynamic operators for agent i executing action a in the former case, and doing so safely in the latter case. Note that the $\mathbf{S}$ in $[\mathbf{i}, a, \mathbf{S}]$ is a constant in the syntax of the logic denoting 'safety', and is not a variable, whereas the $\mathbf{i}, a$ are

variables for agents and actions, respectively. One can read the action operators as "after i executes a, φ is true.' We define the dual modal operators $\langle \mathbf{K}_i \rangle$, $\langle \mathbf{B}_i \rangle$, $\langle \mathbf{i}, a \rangle$, and $\langle \mathbf{i}, a, \mathbf{S} \rangle$ in the usual way:

$$\langle \mathbf{K}_i \rangle \, \varphi \equiv \neg \mathbf{K_i} \, \neg \varphi$$

$$\langle \mathbf{B}_i \rangle \, \varphi \equiv \neg \mathbf{B_i} \, \neg \varphi$$

$$\langle \mathbf{i}, a \rangle \, \varphi \equiv \neg [\mathbf{i}, a] \neg \varphi$$

$$\langle \mathbf{i}, a, \mathbf{S} \rangle \, \varphi \equiv \neg [\mathbf{i}, a, \mathbf{S}] \neg \varphi$$

DASL is axiomatized by the following Hilbert system.

All propositional tautologies are axioms.

$\mathbf{K_i}$ is T (knowledge relation is reflexive)

$\mathbf{B_i}$ is KD45 (belief relation is serial, transitive, and Euclidean)

EP1: $\mathbf{K_i} \, \varphi \Rightarrow \mathbf{B_i} \, \varphi$

EP2: $\mathbf{B_i} \, \varphi \Rightarrow \mathbf{B_i} \, \mathbf{K_i} \, \varphi$

EP3: $\mathbf{B_i} \, \varphi \Rightarrow \mathbf{K_i} \, \mathbf{B_i} \, \varphi$

SP: $[\mathbf{i}, a] \varphi \Rightarrow [\mathbf{i}, a, \mathbf{S}] \varphi$

PR: $\langle \mathbf{i}, a \rangle \, \varphi \Rightarrow \mathbf{B_i} \, \langle \mathbf{i}, a, \mathbf{S} \rangle \, \varphi$,

plus the inference rules Modus Ponens and Necessitation for $\mathbf{K_i}$ and $\mathbf{B_i}$.

DASL allows for formal representation of and reasoning about a pilot's knowledge, belief, actions, and *safe* or *unsafe* actions, and how they all relate to each other logically. In particular, the following theorem is proven:

Theorem 5.1.1 (Unawareness of Safety-Critical Information)**.** *A rational pilot executes an unsafe action only if she is unaware that the action is unsafe.*

The work has its foundations in economics, computer science, and philosophy, where the construction of formal models of reasoning and agent knowledge is of primary concern. DASL improves on the work in those domains for the purposes of practical application to aviation safety by developing a more realistic model of human agents who can be mistaken in their beliefs. The notion of a *rational pilot* in DASL is one who executes an action only if she believes that it is safe, although she may be wrong about this. Merely believing something does not entail knowledge of it. This is critical for our purposes, because we must be able to model agents who are mistaken, but still rational[1].

In DASL, an action is safe or unsafe depending on the conditions under which it is executed. If the conditions for acting are safe, then its *safety precondition* has been met. In the aforementioned work the authors envision the safety precondition of an aviation action as the set of instrument configurations that would permit the action, according to a flight safety manual of the aircraft. Formally, a conjunction of propositions capturing each instrument's value constitutes a *configuration* of the flight deck, *e.g.* "the left airspeed indicator is VERY FAST *and* the right airspeed indicator is SLOW *and* ...", where 'very fast' and 'slow' refer to ranges of the instrument's potential values. The safety precondition function (pre_s) is defined as a mapping from flight control inputs (actions) to the set of configurations under which that input is considered safe according to a flight safety manual.

When the actual configuration of the flight deck conflicts with the safety precondition of an action, executing that action is unsafe. DASL establishes a logical connection between a pilot's actions and her epistemic state, or state of knowledge, through Theorem 5.1.1. The mental state of being *unaware* of φ has the following

[1]This approach to rationality has been taken up recently in the literature with attempts to model agents whose rationality is *bounded*, but by and large, most of the literature treats agents as being perfectly rational.

formal definition:

$$\neg \mathbf{K_i}\, \varphi \wedge \neg \mathbf{K_i}\, \neg \mathbf{K_i}\, \varphi,$$

read as "the pilot i does not know $\boldsymbol{\varphi}$, and i does not know that i does not know $\boldsymbol{\varphi}$," where $\boldsymbol{\varphi}$ is a proposition [**?**]. In epistemic logic, this is identical to a failure of negative introspection [32, 30]. Theorem 5.1.1 establishes that unsafe action from a rational pilot implies a failure of negative introspection about the action's safety precondition. A key upshot from this formal analysis is that by providing negative introspection about the safety precondition, the rational pilot will cease to execute the unsafe action. We prove this in the following corollary.

Theorem 5.1.2 (Awareness of Safety-Critical Information)**.** *If a rational pilot* **i** *has awareness about the safety precondition of action a and the safety precondition is false, then* **i** *does not execute a.*

Proof.

1. $\neg \mathbf{K_i}\, \boldsymbol{pre_s(a)} \Rightarrow \mathbf{K_i}\, \neg \mathbf{K_i}\, \boldsymbol{pre_s(a)}$ ..... Awareness Assumption
2. $\neg \boldsymbol{pre_s(a)}$..... Safety Precondition Assumption
3. $\neg \boldsymbol{pre_s(a)} \Rightarrow \langle \mathbf{i}, \mathbf{a} \rangle\, \boldsymbol{True} \Rightarrow \neg \mathbf{K_i}\, \neg \mathbf{K_i}\, \boldsymbol{pre_s(a)}$..... Theorem 1
4. $\langle \mathbf{i}, \mathbf{a} \rangle\, \boldsymbol{True} \Rightarrow \neg \mathbf{K_i}\, \neg \mathbf{K_i}\, \boldsymbol{pre_s(a)}$..... 2, 3 Modus Ponens
5. $\neg \mathbf{K_i}\, \boldsymbol{pre_s(a)}$..... 2, **K** reflexive
6. $\mathbf{K_i}\, \neg \mathbf{K_i}\, \boldsymbol{pre_s(a)}$..... 1, 5 Modus Ponens
7. $\neg \langle \mathbf{i}, \mathbf{a} \rangle\, \boldsymbol{True}$..... 4, 6 Modus Tollens
8. $[\mathbf{i}, \mathbf{a}]\boldsymbol{False}$..... Def. $\langle \mathbf{i}, \mathbf{a} \rangle$ □

□

Thus, if the countermeasure succeeds in establishing the pilot's awareness of the safety-critical information's being false, it follows by the logic of DASL that the agent discontinues the unsafe action. The goal is to convince her that she does not know what she thinks she knows, like the airspeed, or the horizontal attitude.

The insight gained from the formal analysis allows us to increase aviation safety, but we require a means for computing the safety-critical information that the pilot is unaware of. We do this by encoding the actual configuration of the flight deck and the safety precondition of an input action as a set of constraints and a formula to be examined by a SMT solver, and then check whether it is satisfiable. If it is not, we are interested in retrieving the specific instrument readings that conflict with the safety precondition. In SMT solving, this collection of clauses is called the unsatisfiable core. In the next section, we describe the formal properties of the unsatisfiable core and other ways it is used in formal methods research.

Before providing background on the unsatisfiable core's role in other applications of formal methods, we will provide a bit more discussion on the safety precondition. For this paper, we take it as a given that the safety precondition of an action can be determined by formally representing the contents of flight safety manuals. The specificity of instructions one finds in aviation manuals is one of the principle reasons we have this confidence. Aviation safety frequently discusses the notion of a "flight envelop" as a range of conditions in which safe actions can be executed. In a sense, we are formalizing the flight envelop as constraints for a SMT solver. However, we do not represent the safety preconditions for actions as they would appear in a higher fidelity system, but rather approximate them with a little bit of common sense, in order to achieve a balance between illustration of the approach and brevity. Were this approach to be engineered for real-world applications, the safety preconditions, and indeed the instrument configurations, would no doubt be more complex.

5.2 SMT Solving and the Unsatisfiable Core

Satisfiability Modulo Theories (SMT) is a decision problem from theoretical computer science where a formula from first order logic, constrained by theories in the form of other first order formulas, is assessed for satisfiability. Tools exist to solve engineering problems amenable to formalization in first order logic with constraining theories.

Consider the following example. The first order logic formula is

$$(x < y) \wedge (y = 10) \wedge (x > 20) \wedge (z = 40).$$

The constraining theory is the standard set of axioms of arithmetic. The formula is unsatisfiable.

When a formula constrained by theories is unsatisfiable, the unsatisfiable core refers to the set of subformulas responsible for a clausal formula's being unsatisfiable. That is, the unsatisfiable core of the formula is the portion of it that conflicts with respect either to internal consistency or with the constraining theories. In the example above, the unsatisfiable core is the first three propositions. The unsatisfiable core is successfully applied as a formal method for engineering in the domains of hardware and software verification. When engineers design a chip, before fabricating it they need to verify that the design meets the specifications. One approach to this is to formalize the chip components as variables in a first order formula, and to formalize the specification as a set of constraints on the formula. Then, running the design and specification through a SMT solver, one can identify which aspects of the design are in conflict with the specification. Software is similarly formalized and analyzed during its specification stage.

Some recent work has applied SMT solving to what are known as cyber physical

systems, or hybrid systems [35]. In particular, some hybrid systems are those that involve human machine interaction. Efforts to apply SMT solving to human machine interaction have succeeded in identifying execution paths the system can go down that could cause the human user to become confused. Furthermore, this approach has been applied to the domain of aviation safety. However, as yet no efforts have been undergone to apply SMT solving for the run-time diagnosis and error correction of human error in hybrid systems.

This work does so by leveraging the logical connection between a pilot's action, her knowledge and beliefs about the system, and the features of the system that make the action safe or unsafe. We model the conditions of the action's safe execution as a constraining theory of instrument readings, and the actual instrument readings as a first order formula to be tested for satisfiability modulo that theory. If the formula modeling the actual instrument readings is not satisfiable, we extract the unsatisfiable core. Based on the logical connection established by DASL, we identify the unsatisfiable core with the safety-critical information that the pilot is unaware of when she executes an unsafe action.

5.3 Encoding and Case Studies

We develop an encoding of the instruments and the safety preconditions of actions so that they can be fed into the Z3 SMT solver and, if the actual instrument readings conflict with the constraining safety precondition, their unsatisfiable core may be extracted [**?**]. The case studies we present as examples in this section are meant to illustrate the encoding and the usefulness of the approach. With length limitations, we include three case studies in order to show the versatility of the approach, but sacrifice the complexity of the representations. The actual values of the case studies have

been simplified for presentation, and the encodings themselves have been abridged to include only enough of the instrumentation and safety preconditions so as to illustrate the approach.

In this section we describe the first order logic encoding of actions and their safety preconditions, and present case studies as examples. We begin with Air France 447[2].

5.3.1 Air France 447 Encoded

Air France flight 447 from Rio de Janeiro, Brazil, to Paris, France, occurred June 1, 2009. The Airbus A330 encountered adverse weather over the Atlantic Ocean, resulting in a clogged Pitot-static system. Consequently, the airspeed indicators delivered unreliable data to the pilot flying, resulting in confusion. A chain of events transpired in which the pilot overcorrected the plane's horizontal attitude again and again, and continued to input nose up pitch commands, all while losing airspeed. Perhaps most confusing to the pilot was the following situation: the aircraft's angle of attack (AOA) was so high it was considered invalid by the computer, so no stall warning sounded until the nose pitched down into the valid AOA range, at which point the stall warning would sound. When the pilot pulled up, the AOA would be considered invalid again, and the stall warning would cease. The aircraft entered a spin and crashed into the ocean. Palmer [42] argues that had the pilot merely taken no action, the Pitot tubes would have cleared in a matter of seconds, and the autopilot could have returned to Normal Mode. The official Bureau d'Enquêtes et d'Analyses pour la Sécurité de l'Aviation Civile (BEA) investigation found that a procedure for safely dealing with unknown airspeeds was known to the pilots but not engaged in, and this may have saved the flight as well.

For this case study, we encode the segment of the mishap chain concerning the

[2]The source code for the case studies can be found here: https://github.com/sethahrenbach/UnsatCore

inconsistent airspeed indicator readings and the hard nose up pitch command that the pilot continuously executed. We begin by declaring the variables of the scenario.

```
;; Air France 447 Case
(set-option :produce-unsat-cores true)

;; right side instruments
(declare-const right-airspeed Int)
(declare-const right-pitch Int)
(declare-const right-bank Int)

;; middle instruments
(declare-const middle-airspeed Int)
(declare-const middle-pitch Int)
(declare-const middle-bank Int)

;; left side instruments
(declare-const left-airspeed Int)
(declare-const left-pitch Int)
(declare-const left-bank Int)
```

Next we encode the actual instrument readings during the flight. We give these clauses names, so that the unsatisfiable core extraction can refer to them if they are responsible for a logical conflict.

```
;; actual instrument readings
(assert (!(= left-airspeed 135) :named lair))
(assert (!(= right-airspeed 100) :named rair))
(assert (!(= middle-airspeed 100) :named mair))
(assert (!(= right-pitch 15) :named rpitch))
(assert (!(= middle-pitch 15) :named mpitch))
(assert (!(= left-pitch 15) :named lpitch))
(assert (!(= right-bank 15) :named rbank))
(assert (!(= middle-bank 15) :named mbank))
(assert (!(= left-bank 15) :named lbank))
```

Next, we encode the safety precondition of the hard nose up input action. We capture the requirement that a hard nose up should not be engaged in if the indicators for the same aspect of flight, like airspeed, are in disagreement. We also capture some upper and lower bounds between which the indicators must read in order for the input to be considered safe. Normally, these bounds are thought of as the flight envelope.

```
;; safety precondition of action: Hard Nose Up
(assert (= left-airspeed right-airspeed))
(assert (= left-airspeed middle-airspeed))
(assert (= middle-airspeed right-airspeed))
(assert (= right-pitch left-pitch))
(assert (= right-pitch middle-pitch))
(assert (= middle-pitch left-pitch))
(assert (< right-airspeed 700))
(assert (> right-airspeed 99))
(assert (< middle-airspeed 700))
(assert (> middle-airspeed 99))
(assert (< left-airspeed 700))
(assert (> left-airspeed 99))

(check-sat)
(get-unsat-core)
```

The final commands of the .smt2 file tell Z3 to check whether the formula is satisfiable, and if not, to retrieve and print the unsatisfiable core. The command line input and output for running Z3 on the file appear below.

```
input : z3 af447.smt2
output: unsat
output: (lair rair)
```

After downloading the Z3 executable[3] and creating the af447.smt2 file, we run Z3

[3]https://github.com/Z3Prover/z3/releases

on the file in a single command line input, and receive two output lines. One line tells us that the formula is unsatisfiable, which means that there is some conflict between the actual instrument readings and the safety precondition for the action. The second line tells us which instrument readings are responsible for the conflict. In this case, the instrument readings ***lair*** and ***rair*** are determined to be the conflict. If we look up at the file contents, we see that the clauses with those labels capture the left airspeed indicator, reading 135, and the right airspeed indicator, reading 100. According to the safety precondition, these values should be equal. Otherwise, a hard nose up input is unsafe, because the airspeed is not known.

The unsatisfiable core of the formula, as extracted by the Z3 SMT solver, infers the safety-critical information of the case, because it is that pair of instrument readings that render the safety precondition false, by contradicting the condition that they be in agreement.

At this point, one might fairly wonder why ***mair*** is not likewise included in the set. This is because it is not necessary for identifying the source of the contradiction. Having found that ***lair*** and ***rair*** conflict, these suffice to serve as the unsatisfiable core. And indeed, it is irrelevant whether the core is identified as ***lair*** and ***rair*** or ***lair*** and ***mair***, because either will adequately serve as the safety critical information. Both sufficiently demonstrate a conflict in the relevant indicators.

5.3.2 Copa Flight 201 Encoded

Copa flight 201 departed Panama City, Panama for Cali, Colombia in June, 1992 [**?**]. Due to faulty wiring in the captain's Attitude Indicator, he incorrectly believed he was in a left bank position. In response to this, he directed the plane into an 80 degree roll to the right, which caused the plane to enter a steep rolling dive. At some point one of the pilots switched the input to the backup Attitude Indicator to be that of the pilot's faulty one, furthering confusion. Approximately 29 minutes after

takeoff, the plane crashed into the jungle and all passengers and crew perished. We formalize the moment at which the pilot provides the hard right roll input.

We encode the variables the same as before, and leave out the code to save space. A simplification of the actual instrument readings is encoded as follows.

```
;; actual instrument readings
(assert (!(= left-airspeed 135) :named lair))
(assert (!(= right-airspeed 135) :named rair))
(assert (!(= middle-airspeed 135) :named mair))
(assert (!(= right-pitch 15) :named rpitch))
(assert (!(= middle-pitch 15) :named mpitch))
(assert (!(= left-pitch 15) :named lpitch))
(assert (!(= right-bank -15) :named rbank))
(assert (!(= middle-bank 0) :named mbank))
(assert (!(= left-bank 0) :named lbank))
```

Then the safety precondition for a hard right bank would include the condition that the instrument readings agree on the plane's actual bank position.

```
;;safety precondition of action: Hard Right Bank
(assert (= left-airspeed right-airspeed))
(assert (= left-airspeed middle-airspeed))
(assert (= middle-airspeed right-airspeed))
(assert (= right-bank left-bank))
(assert (= right-bank middle-bank))
(assert (= middle-bank left-bank))
(assert (< right-airspeed 700))
(assert (> right-airspeed 99))
(assert (< middle-airspeed 700))
(assert (> middle-airspeed 99))
(assert (< left-airspeed 700))
(assert (> left-airspeed 99))

(check-sat)
(get-unsat-core)
```

When we save the file as copa201.smt2 and run Z3 on it, we see the following in the terminal.

```
input : z3 copa201.smt2
output: unsat
output: (rbank lbank)
```

Again, as before, the Z3 SMT solver allows us to infer which safety-critical information in particular the pilot is unaware of, because it is the information that conflicts with the safety precondition. What the pilot's action reveals is that he mistakenly believed,

at least at first, that he was in a steep left bank, because he was relying on the faulty instrument in front of him. Had he grasped the safety-critical information, that the attitude indicators were in stark disagreement, he would have realized that he did not know the plane's bank position. In this case, safety procedure is to maintain a level horizontal input, absent other convincing evidence that a steep left bank is occurring, which there was not in this case. Before executing such an extreme action, proper cross-checking of other instruments must occur.

5.3.3 Asiana Flight 214 Encoded

Asiana flight 214 from South Korea to San Francisco departed in the evening of July 6, 2013 and was scheduled to land just before noon that morning [41]. Air traffic control cleared the pilots to perform a visual approach to the runway. The plane came in short and crashed against an embankment in front of the runway, resulting in the deaths of three passengers and 187 injured. The National Transportation Safety Board (NTSB) investigation found that the captain had mismanaged the approach and monitoring of the airspeed, resulting in the plane being too high for a landing. When the pilot realized the plane's glide path was too high, he accidentally disconnected the autopilot's monitoring of the airspeed. This caused an autothrottle (A/T) protection to turn off, so when he pitched the nose down, the plane descended faster than was safe, causing it to collide with the embankment in front of the runway. We will formalize the moment at which the pilot pitches the nose down, unaware that the autothrottle is turned off.

In encoding this state of affairs, we add a new global state variable to capture the autothrottle status.

```
;; Asiana 214 Case
(set-option :produce-unsat-cores true)

;; global state
(declare-const autothrottle Bool)
```

We likewise include right, middle, and left instruments for altitude in the variable declarations, omitted here for space. The encoding of the actual instrument readings is as follows.

```
;; actual instrument readings
(assert (!(= autothrottle false) :named at-status))
(assert (!(= right-alt 5000) :named ralt))
(assert (!(= middle-alt 5000) :named malt))
(assert (!(= left-alt 5000) :named lalt))

(assert (!(= left-airspeed 200) :named lair))
(assert (!(= right-airspeed 200) :named rair))
(assert (!(= middle-airspeed 200) :named mair))
```

Finally, for the action of a Hard Nose Down, we show how to encode a conditional constraint, that if the altitude is less than 10000 feet, then the autothrottle must be on in order for a Hard Nose Down to safely be executed. Logically, this conditional is equivalent to the proposition that either the altitude is greater than (or equal to, but we omit this in the formalization) 10000 feet, or the autothrottle is on.

```
;; safety precondition of action: Hard Nose Down
(assert (= left-alt right-alt))
(assert (= left-alt middle-alt))
(assert (= middle-alt right-alt))
(assert (= left-airspeed right-airspeed))
(assert (= left-airspeed middle-airspeed))
(assert (= middle-airspeed right-airspeed))
(assert (or (> right-alt 10000) (= autothrottle true)))



(check-sat)
(get-unsat-core)
```

When we save the file as asiana.smt2 and run Z3 on it, we see the following in the terminal.

```
input : z3 asiana.smt2
output: unsat
output: (at-status ralt)
```

This indicates that the safety-critical information is the combination of the altitude, here Z3 returns only the right side altimeter reading because it is sufficient, and the fact that the autothrottle is off, just as the NTSB report concluded.

5.4 The Unsatisfiable Core is the Safety-Critical Information

This section establishes the identification of the unsatisfiable core extracted from Z3 with the safety-critical information that a pilot is unaware of when she executes an unsafe action. To do this, we examine a case study formalized in DASL, and compare the results with the same case study encoded as an SMT problem that was run through Z3.

Consider the following formalization of Copa flight 201 in DASL. Line 1 represents a conjunction over the actual instrument readings, encoded as atomic propositions. The relevant propositions included in the text state that the right attitude indicator (*RightBank*) reads that the aircraft is pitching left considerably (*SteepLeft*), but that the left attitude indicator (*LeftBank*) reads level flight. We ignore the middle attitude indicator to try and keep it readable, but if it were included, it would be present in the conclusion as well. Line 2 expresses in DASL that the pilot executes a hard right bank action. Line 3 concludes with the theorem's application, which states that the pilot lacks awareness either the left indicator or the right indicator: she is unaware of the falsity of one of them. This analysis establishes that the attitude indicators' values together are a piece of safety-critical information, because it shows that the pilot engages in the unsafe action *only if* she remains unaware of them jointly.

1. $(\boldsymbol{RightBankSteepLeft}) \wedge (\boldsymbol{LeftBankLevel}) \ldots\ldots\ldots$ configuration.

2. $\langle \boldsymbol{pilot}, \boldsymbol{hardrightbank} \rangle\ \boldsymbol{true} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$ pilot input.

3. $\neg \mathbf{K}_{pilot}\ (\boldsymbol{LeftBankSteepLeft} \vee \boldsymbol{RightBankLevel}) \wedge$
   $\neg \mathbf{K}_{pilot}\ \neg \mathbf{K}_{pilot}\ (\boldsymbol{LeftBankSteepLeft} \vee \boldsymbol{RightBankLevel}) \ldots\ldots$ 1,2,Th 1.

Compare this with the results of running the SMT encoding through Z3. The instrument readings construed as a first order forumla are unsatsifiable modulo the constraints imposed by the action's safety precondition. The unsatisfiable core identified by Z3, that is, the reason it is unsatisfiable, is because the right attitude indicator (***rbank***) and the left attitude indicator (***lbank***) are in disagreement. This is consistent with the DASL analysis, which identifies the left attitude indicator's value as the one the pilot is critically unaware of.

The way these two formalisms work together is that the DASL theorem establishes the connection between the action, pilot knowledge, and safety conditions, and the SMT solver performs the safety-critical inference in an efficient and expressive way. They are different in an interesting way, though. DASL works by identifying a proposition from the safety precondition of the action that is *false*, not currently present on the instrument panel. For example, a hard right bank action's safe execution requires the attitude indicators to be in agreement. If the attitude indicators disagree, and the pilot executes the action anyway, then DASL allows us to infer unawareness of the false propositions from the safety precondition. The SMT formalization operates by identifying which of the *true* propositions from the instrument panel conflict with the safety precondition of the action. Thus, the result of DASL: the pilot is unaware

regarding the left indicator's being in a steep left bank or the right indicator's being level; Z3 infers that the actual instrument readings of the right airspeed indicator and the left airspeed indicator are the ones that the pilot needs to see. We built it this way so that the instruments themselves, not the safety precondition, are the output of the Z3 engine, for reasons to be discussed in the next section.

5.5 Application Design

This section briefly describes how this encoding of aviation activity and the Z3 theorem prover might be used in a runtime monitor application.

The application would execute the following algorithm in a loop.

Algorithm 1 Monitor Algorithm

```
while flying do
    smt2_actual_readings ← instrument readings
    smt2_safety_preconditions ← lookup_safety_precondition(flight control input)
    smt2_file ← concat(smt2_actual_readings, smt2_safety_precondtions)
    unsat_core ← Z3 smt2_file
    if unsat_core is empty then undim all
    else if unsat_core is not empty then
        map(dim, [x|x ∉ unsat_core])
```

This algorithm, though simple, represents a dramatic departure from contemporary safety practices, in that it aims to draw the pilot's attention to the safety-critical information by *dimming* the information that is not, at the moment, safety-critical. Dimming, of course, is not a technical term, and has not been defined. Its definition and implementation depends on the context and the technical details of the cockpit.

In most emergency situations, a number of alarms are going off at once, and the pilots suffer from information overload, which decreases their situational awareness. The safety-critical information is right in front of them, and perhaps even has an alarm and blinking light accompanying it. And yet they still crash. This algorithm

addresses the problem by temporarily *removing* the alarms and blinking lights that are not related to the safety-critical information identified by Z3's returned unsatisfiable core, based on the encoding of the configuration and action. If the only alarm or blinking lights are those highlighting the fact that, say, the airspeed indicators are in disagreement, immediately following the pilot's action to dramatically reduce thrust, then the chances that he or she notices this problem must increase.

This runtime monitor will increase the likelihood that the pilot notices the safety-critical information. Noticing it, he or she will cease the unsafe action.

5.6 Related Work

Over the past decade related work in applying formal methods to the human component of aviation safety have advanced the state of the art and contributed to aviation safety [35, 21, 17, 14]. However, previous work applies formal methods to the specification stage of systems, typically using model checking or SAT/SMT solving to discover possible states of the system in which a human might become disoriented or confused. This work is important. The present work advances a different but related line of effort, applying formal methods to the run-time stage of systems, with the goal of diagnosing and correcting unsafe system behaviors as they occur, especially those caused by human confusion. To do this, we have taken a formal model of the pilot's reasoning in the form of DASL and used the resulting analysis to develop a tool for delivering safety-critical information to the pilot when it is needed.

Chapter 6

Summary and concluding remarks

Congratulations on completing your dissertation.

Appendix A

Title of first appendix

A.1 Section title

Here is some additional information which would have detracted from the point being made in the main article.

A.1.1 Subsection title

This section even has subtitles

Bibliography

[1] Federal Aviation Administration
website: https://www.faasafety.gov/files/gslac/library
/documents/2012/Nov/71574/DirtyDozenWeb3.pdf,
retrieved 10/5/2016.

[2] Rankin, William. "MEDA Investigation Process", Aero Quarterly.
Website: Boeing.com/commercial/aeromagazine/articles/qtr_2_07/aero_q207_article3.pdf,
retrieved 10/5/2016

[3] Aviation Maintenance Technician Handbook. Chapter 14, "Addendum/Human Factors".

[4] van Ditmarsch, H.; van der Hoek, W.; Kooi, B. Dynamic Epistemic Logic. Springer. Dordrecht, The Netherlands, 2008.

[5] Ahrenbach, S.; Goodloe, A. Formal Analysis of Pilot Error Using Agent Safety Logic. *Innovations in Systems and Software Engineering.* **Submitted**.

[6] Bolton, M.; Bass, E.; Siminiceanu, R. Using formal verification to evaluate human-automation interaction: A review. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on* (Vol 43, No. 5, pp.488-503). 2013. IEEE.

[7] Bolton, M.; Bass, E. Formally verifying human–automation interaction as part of a system model: limitations and tradeoffs. *Innovations in systems and software engineering* (Vol 6, No. 3, pp.419-231). 2010. Springer-Verlag.

[8] Bolton, M.; Bass, E.; Siminiceanu, R. A systematic approach to model checking human–automation interaction using task-analytic models. *Systems, Man, and Cybernetics: Systems and Humans, IEEE Transactions on* (Vol 41, No. 5, pp.961-976). 2011. IEEE.

[9] Barras, B.; Boutin, S.; Cornes, C.; Courant, J.; Filliatre, J.C.; Gimenez, E.; Herbelin, H.; Huet, G.; Munoz, C.; Murthy, C. and Parent, C. The Coq proof assistant reference manual: Version 6.1 (Doctoral dissertation, Inria). 1997.

[10] Blackburn, P.; de Rijke, M.; Venema, Y. Modal Logic. Cambridge University Press. New York, 2001.

[11] Baltag, A.; Moss, L.; Solecki, S. "The logic of public announcements, common knowledge, and private suspicions." Readings in Formal Epistemology. Springer International Publishing, 2016. 773-812.

[12] Fagin, R.; Halpern J.; Moses, Y.; Vardi, M. Reasoning about knowledge. MIT press, 2004.

[13] Basin, D., Radomirovic, S. and Schmid, L., Modeling Human Errors in Security Protocols.

[14] Butler, R.W., Miller, S.P., Potts, J.N. and Carreno, V.A., 1998. A formal methods approach to the analysis of mode confusion. In Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE (Vol. 1, pp. C41-1). IEEE.

[15] Butler, R.; Johnson, S. Formal Methods For Life-Critical Software. *Computing in Aerospace 9 Conference*, (pp. 319-329), San Diego, CA. October 1993.

[16] Davis, M.; Putnam, H. "A Computing Procedure for Quantification Theory". J.ACM. 7 (3): 201–215. (1960).

[17] Rushby, J., 1993. Formal methods and the certification of critical systems. SRI International. Computer Science Laboratory.

[18] Aucher, G. and Schwarzentruber, F., 2013. On the complexity of dynamic epistemic logic. arXiv preprint arXiv:1310.6406.

[19] et d'Analyses, Bureau dEnqutes. "Final report on the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro–Paris." Paris: BEA (2012).

[20] van Benthem, J.; Pacuit, E; Roy, O. Toward a Theory of Play: A Logical Perspective on Games and Interaction. *GAMES* **2011**, *2*, 52-86.

[21] Rushby, J. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety* **2002**, *75*, 167-177.

[22] Rushby, J. On the Interpretation of Assurance Case Arguments. *Proceedings of the Second International Workshop on Argument for Agreement and Assurance (AAA 2015)*. Springer LNCS.

[23] Rushby, J. Trustworthy Self-Integrating Systems. *Springer LNCS*, **2016**, *9581*, 19-29.

[24] Gelman, G.; Feigh, K.; Rushby, J. Example of a Complementary use of Model Checking and Agent-based Simulation. *IEEE Int'l Conf. on Systems, Man, and Cybernetics* **2013**, 900-905.

[25] Baltag, A.; Moss, L.; Solecki, S. The Logic of Public Announcements, Common Knowledge, and Private Suspicions. *TARK '98*, **1998**, 43-56.

[26] Shoham, Y.; Leyton-Brown, K. *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundation*. Cambridge University Press: New York, USA, 2008.

[27] van Ditmarsch, H.P.; van der Hoek, W.; Kooi, B.P. Dynamic Epistemic Logic with Assignment. *Proceedings of the fourth internal joint conference on Autonomous agents and multiagent systems*, **2005**, 141-148.

[28] van Benthem, J. *Logical Dynamics of Information and Interaction*. Cambridge University Press: New York, USA, 2011.

[29] Van Ditmarsch, H. "Knowledge games." Bulletin of Economic Research 53.4 (2001): 249-273.

[30] Fagin, R.; Halpern J.; Moses, Y.; Vardi, M. *Reasoning About Knowledge*. The MIT Press: Cambridge, USA, 2003.

[31] Harel, D.; Kozen, D. and Jerzy Tiuryn. Dynamic logic. MIT press, 2000.

[32] Hintikka, J. *Knowledge and Belief*. Cornell University Press. Ithaca, USA, 1962.

[33] Barwise, J.; Seligman, J. *Information Flow: The Logic of Distributed Systems*. Cambridge University Press: New York, USA, 1997.

[34] van Ditmarsch, H.; van der Hoek, W.; Kooi, B. *Dynamic Epistemic Logic*. Springer: Dordrecht, The Netherlands, 2008.

[35] Bass, E.; Feigh K.; Gunter, E.; Rushby, J. Formal Modeling and Analysis for Interactive Hybrid Systems. Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems (FMIS 2011). Electronic Communications of the EASST. Vol 45 (2011).

[36] Rushby, J. Formalism in Safety Cases. In *Making Systems Safer: Proceedings of the 18th Safety-Critical Systems Symposium*; Dale, C.; Anderson, T.; Eds. Bristol UK, 2010, 3-17.

[37] Klir, G.; Yuan, B. Fuzzy sets and fuzzy logic. Vol. 4. New Jersey: Prentice hall, 1995.

[38] Lescanne, P. "Mechanizing common knowledge logic using COQ." Annals of Mathematics and Artificial Intelligence 48.1-2 (2006): 15-43. APA

[39] De Moura, L.; Bjørner, N. Z3: An efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems*. 2008. Springer.

[40] Maliković, M.; Čubrilo, M. "Modeling epistemic actions in dynamic epistemic logic using Coq." CECIIS-2010. 2010.

[41] Maliković, M.; Čubrilo, M. "Reasoning about Epistemic Actions and Knowledge in Multi-agent Systems using Coq." Computer Technology and Application 2.8 (2011): 616-627.

[42] National Transportation Safety Board. "Descent below visual glidepath and impact with Seawall Asiana Flight 214, Boeing 777-200ER, HL 7742, San Francisco, California, July 6, 2013 (Aircraft Accident Report NTSB/AAR-14/01)." Washington, DC Author: NTSB (2014).

[43] Palmer, Bill. Understanding Air France 447. William Palmer, 2013.

[44] Lescanne, P.; Puisségur, J. "Dynamic logic of common knowledge in a proof assistant." arXiv preprint arXiv:0712.3146 (2007).

[45] Aumann, R. "Interactive epistemology I: knowledge." International Journal of Game Theory 28.3 (1999): 263-300.

[46] Hwang, M.; Lin, J. "Information dimension, information overload and decision quality." Journal of Informatin Science 25 (1999): 213-218.

[47] Simpson, C.; Prusak, L. "Troubles with Information Overload = Moving from Quantity to Quality in Information Provision." International Journal of Information Management 15 (1995): 413-425.

VITA

This is a summary of your *professional* life, and should be written appropriately. This can be written in the following order: where your where born, what undergraduate university you graduated from, if you received a masters, and which institution you graduated from with your PhD (University of Missouri). You can describe when you began research with your current advisor.

In another paragraph, you could say if/when you were married, what the name of your kids are, and what your plans are for after graduation if you choose. Take a look at other vita's from other dissertations for examples.