FORMALIZING OPERATOR TASK ANALYSIS

BY

AYESHA YASMEEN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Abstract

Human operators are unique in their decision making capability, judgment and nondeterminism. Their sense of judgment, unpredictable decision procedures, susceptibility to environmental elements can cause them to erroneously execute a given task description to operate a computer system. Usually, a computer system is protected against some erroneous human behaviors by having necessary safeguard mechanisms in place. But some erroneous human operator behaviors can lead to severe or even fatal consequences especially in safety critical systems. A generalized methodology that can allow modeling and analyzing the interactions between computer systems and human operators where the operators are allowed to deviate from their prescribed behaviors will provide a formal understanding of the robustness of a computer system against possible aberrant behaviors by its human operators.

We provide several methodology for assisting in modeling and analyzing human behaviors exhibited while operating computer systems. Every human operator is usually given a specific recommended set of guidelines for operating a system. We first present process algebraic methodology for modeling and verifying recommended human task execution behavior. We present how one can perform runtime monitoring of a computer system being operated by a human operator for checking violation of temporal safety properties.

We consider the concept of a protection envelope giving a wider class of behaviors than those strictly prescribed by a human task that can be tolerated by a system. We then provide a framework for determining whether a computer system can maintain its guarantees if the human operators operate within their protection envelopes. This framework also helps to determine the robustness of the computer system under weakening of the protection envelopes. In this regard, we present a tool called Tutela that assists in implementing the framework.

We then examine the ability of a system to remain safe under broad classes of variations of the prescribed human task. We develop a framework for addressing two issues. The first issue is: given a human task specification and a protection envelope, will the protection envelope properties still hold under standard erroneous executions of that task by the human operators? In other words how robust is the protection envelope? The second issue is: in the absence of a protection envelope, can we approximate a protection envelope encompassing those standard erroneous human behaviors that can be safely endured by the system? We present an extension of Tutela that implements this framework.

The two frameworks mentioned above use Concurrent Game Structures (CGS) as models for both computer systems and their human operators. However, there are some shortcomings of this formalism for our uses. We add incomplete information concepts in CGSs to achieve better modularity for the players. We introduce nondeterminism in both the transition system and strategies of players and in the modeling of human operators and computer systems. Nondeterministic action strategies for players in *i*ncomplete information *N*ondeterministic CGS (iNCGS) is a more precise formalism for modeling human behaviors exhibited while operating a computer system. We show how we can reason about a human behavior satisfying a guarantee by providing a semantics of Alternating Time Temporal Logic based on iNCGS player strategies. In a nutshell this dissertation provides formal methodology for modeling and analyzing system robustness against both expected and erroneous human operator behaviors.

To my parents: Zainul and Aklima Abedin

Acknowledgments

I first acknowledge the contributions of my advisor Professor Elsa L. Gunter. This dissertation would not have been possible without her relentless support throughout my years at UIUC. Her eagerness and patience in teaching me the topics that I was ignorant about was in my opinion phenomenal. Her mathematical analysis prowess has never failed to impress me. I must also mention her eagerness to help and understand the problems of a student mother. I will never forget the days she allowed me to bring in my sick daughter with me to meet with her. She has introduced my eldest daughter to puzzles, played with my younger daughter, offered to teach me driving, lent me a sturdy brush to clean up my rented house, loaned me quarters for the parking meters whenever I was out, inspired me to volunteer at my daughter's school, introduced me to Lego Robotics, and above all shown how to be a wonderful mother and successful Professor and researcher simultaneously.

I thank Professor Carl Gunter for introducing me to the topic of human operator influence on system operations. I have always found his comments and suggestion invaluable. I thank Professor Grigore Roşu for suggesting runtime monitoring to me. I thank Professor Darko Marinov for suggesting applying testing techniques to my research. I thank Professor Ralph Johnson for his realistic, informative comments. I am very grateful to Professor Lori Clarke for her elaborate comments, suggestions about both my research and venues for publication.

I also thank my parents Zainul and Aklima Abedin. They have loved and supported me unconditionally throughout my life. My greatest regret is not being able to complete this dissertation before my father left us for good.

I thank my husband Monir Mozumder for relentlessly supporting me and helping me with taking care of my elder daughter at first and my younger daughter since last year. I would not have come this far without his constant inspiration and unwavering support. I thank my daughters Nuha Mozumder and Alisha Mozumder for providing me with the urge to pursue and finish my goal at UIUC through their innocent smiles.

I also thank my colleagues Andrei Popescu, Christopher Osborn, Majid Kazemian for being so full of encouragement and support. I also thank Dr. Marsha Woodbury and Professor Margaret Fleck for being such wonderful and understanding instructors in the courses I have served as a TA. I also thank Trisha Benson, Donna Coleman, Barbara Leisner, Shirley Finke, Barbara Cicone, Donna Kennedy, and of course Mary Beth Kelly. Their support and help made academic life, getting paychecks, conference travel, and dissertation submission so much easier.

Table of Contents

List of Figures

Chapter 1

Introduction

More than two thousand years ago, humans felt the need for a machine to aid in performing mathematical computations. Then they invented the *abacus*. The realization of the necessity of computational tools and the endeavors to make them a reality can be observed across centuries through the invention of Napier's Bones, Leonardo da Vinci's drawings, Pascal's calculator *Pascaline*, Leibnitz's calculator, Babbage's Analytical Engine, and in the last century the Hardavrd Mark I, the ENIAC, the ILLIAC, the UNIVAC, the IBM mainframes, and finally the personal computers. The revolutionary advances in the computational power, compactness, availability and affordability have made computers an integral part of our lives.

In today's world humans interact with computers in almost every sector of life: through using personal computers at home; using self check-in kiosks at airports and clinics; taking automated standardized tests for driver licenses and admission to educational institutions; withdrawing money from a bank's ATM; and so on. Although computers are used to simplify human lives, the humans themselves may willingly or unwillingly complicate the safety and dependability of the computer systems by adversely influencing their operations. They can jeopardize the operation of a computerized system by using the computers in a manner that is unexpected by the designers of the system. Just as human operators can help maintain the safety of a system by their capabilities of perception, diagnosis, and problem-solving techniques, they can introduce operational errors due to their unpredictability, judgment, and action improvisation tendencies. Computer systems are developed to behave in a predictable and repeatable manner. Their interaction patterns can be anticipated by the operators operating them. Human operators on the other hand are highly unpredictable in nature. Each human operator's behavior can be unique based on their experience, expertise, personality and current frame of mind. Even the same operator may behave differently at different times. Often deviant human operator actions can pose an unintentional threat to a computer system. Erroneous human operator behavior can be hazardous for safety critical systems like aircrafts and air traffic control systems, medical treatment systems and nuclear reactors. From the infamous "Three Mile Island accident" [65] to patients dying from wrong doses of radiation [75], human handling and judgment has led to immense and in some cases fatal consequences. Hence computer systems need to be not only user-friendly but also user-safe. The impacts of failures of the computer systems themselves in our personal, social, economic, national, and global lives are an extensively studied topic. Formal verification of computer systems is thus a well established research area.

Let us consider the Three Mile Island Accident. The incident started to take shape on March 28, 1979, in the Three Mile Island Nuclear Generation Station with failures in the non-nuclear secondary system. This failure was followed by a valve getting stuck open leading to draining of vital nuclear reactor coolant. There was an indicator light to warn the operators about the stuck open valve. However, the design of the indicator was ambiguous. When everything operates smoothly, the indicator light remains turned off indicating that the valve is closed. However, when an unusual condition happens, the light still remained off indicating that the valve is open. The reactor operators were not trained properly about this ambiguous attitude of the indicator light. They misunderstood the situation and thought that there were an *excess* of coolants and took steps to remove coolants from the reactor. They actually needed to ensure pumping in more coolants to maintain safe temperature. The temperature inside the reactor kept rising. There was a temperature indicator capable of displaying the abnormally high temperature. But this indicator was in the back of the panel hidden from the operators and thus virtually useless. In this scenario the operators were not adversaries but allies willing to ensure the safe operation of the generating station. However, human factors combined with insufficient training made the human operators unintentionally fail to follow necessary steps and led to a partial meltdown of the reactor.

The significance of human task execution behaviors have led to the research on *Operator Task Analysis (OTA)* [72, 67, 104, 42]. OTA combines ideas from Human Factors (HF) and Human Computer Interactions (HCI) to analyze what the expected human operator behavior while executing a task is. In order to determine possible errors and therefore increase robustness of the system. We augment this line of investigation by providing a formal methodology for modeling and analyzing how the manner in which an operator performs a task affects the safe and effective operation of a system. We will not consider the cognitive aspects of human decision making, instead we will focus on all possible human operator action sequences and their effects. Moreover, we will focus on the task specifications for operating computerized systems.

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

In systems depending heavily on human operators, providing tolerance of human errors becomes more critical. Fault tolerance analysis considers tolerance of both hardware and software failure and malfunctions. We intend to complement that research by focusing on an area that is often either ignored or oversimplified. That area is the uncertainty contributed by deviations of operators from procedures recommended to them. Human operators have the capability to subjectively introduce errors in the system. They may unpredictably vary their behavior based on frame of mind, time, presence and influence of other operators etc. Current formal frameworks sometimes omit any assumptions about the human operators, sometimes they are considered as all powerful antagonists who can do absolutely anything, and at some other times they are assumed to behave perfectly and always operate within the guidelines provided to them. At some other times, a generic set of assumptions are made about the entire environment in general. We suggest providing a middle ground, where the human operators are neither neglected, nor are they allowed to be unrealistically powerful antagonists or unrealistically perfect operators. The human operators are also not considered some random part of the environment. Instead the human operators can deviate within "tolerable" bounds from their recommended task execution behaviors. From aircraft pilots to smart warehouse operators, in all scenarios the human operators are given specific recommended task descriptions to follow. Of course the recommended task specifications themselves need to be correct and the bulk of research activity focuses on proving the safety and effectiveness of recommended task specifications. The focus of this dissertation is identifying, modeling, verifying and analyzing the tolerable bounds of atypical human task execution behavior. Every human interaction-intensive system will benefit from being modeled and analyzed against variations of human task execution manners.

A positive impact that we intend for our work to have is that, in the case where a possible human error can be identified as a source of threat to a system, the system developers will get interested enough that they will attempt to prevent such an aberrant behavior from occurring in the first place. They will investigate the possible reasons for aberrant human behavior. If there is a lack of proper guidelines or recommended behavior they will try to improve upon them; if there is ambiguous human-computer interface they may either improve the interface or provide rigorous training to eradicate any confusion to disambiguate it. Consider the Three Mile Island Accident. There the expectation was that in the case of a valve problem, the operators will take immediate steps to seal the reactor so that coolants do not leak. This should have been part of the recommended behavior for the operators in case of an emergency. However, in this case ambiguous display system caused the operator to omit the action of containing the coolants. The consequences of such a system is of such enormous proportions that the system designers could have been encouraged by a formal analysis delineating the dangers of the human operators not behaving in the expected manner. They could have incorporated measures to ensure safety like: (i) detecting that operators have failed to perform leak containment actions, (ii) some unambiguous secondary alert is set off that is extremely hard to ignore or (iii) some drastic secondary safety measure is built into the system to contain coolants.

A system may not be safe against all possible human action variations. Our analysis will provide the designers with a comprehensive list of human action variations that can be harmful to the system. The designers can now make informed decisions about which action variations can be or should be reconsidered and possibly be made safe against.

This dissertation demonstrates that:

A formal understanding of the tolerance of the system of both desired and erroneous human behaviors while operating that system can be provided.

In order to support this claim, we first present a methodology for formulating, modeling and verifying the desired or recommended manner of task execution by human operators. The questions that we answer are: Is there a way to model and analyze computer systems as they get operated by humans behaving in the recommended manner? What is the best way to express system requirements and human behavior guarantees?

Then we explore dynamic analysis techniques to study safety conformance of a system combined with a human operator. Dynamic analysis studies execution of a software while interacting with a real or virtual set of inputs. Unlike static verification of computer systems, where each and every trace should conform to a given property for it to satisfy that property, in dynamic analysis we focus on one particular trace of execution. The question we answer is: can we monitor a computer system while interacting with a specific human operator behavior? Then we present a methodology to dynamically monitor a system while interacting with *a* human behavior to determine whether the system is robust against it.

We then move on to atypical human task execution behaviors and move back to static verification. We define "tolerable" sets of deviant human operator behavior. The questions we answer are: Is there a way for modeling human operators and computer system components as they interact with each other, which is suitable for model checking? Is there a way to model the recommended task execution behavior? Is there a way to encode the set of human behaviors which we are assuming the system to be robust against? Is there a way to verify whether the system is indeed robust against this set of atypical human operator behaviors? Is there a simple way of checking the system robustness against various human behaviors guarantees? We then present a prototype tool that helps implement our answers. We also present how we can determine the human operator or system component that leads to requirement violation using their guarantees of "tolerable behaviors".

Then we present methodology for assessing the robustness of the "tolerable" set of human operator behavior itself. The question we answer is: Does the set of "tolerable" behaviors contain well-established variations of human task executions manner like: repetition, out-of-order execution etc. ? If it is hard to ascertain the set of the "tolerable" behaviors, can we suggest possible such behaviors? We then present an extension of our prototype tool that implements our answers to these questions.

Finally, we refocus on the actual model chosen for formalizing computer systems and human operators. We present the need for a more realistic model for encoding the human operators and the system components. Then we present improved formalisms for modeling human behavior and show how to analyze the human behaviors expressed using them.

1.1 Research Contributions

This dissertation has the following contributions:

- Methodology based on process algebra for modeling recommended human operator task execution behavior.
  - We provide a uniform modeling technique for computer systems, and their human operators using Communicating Sequential Processes (CSP) [56, 99].
  - We show how the combined interactions of systems and humans can be verified for requirement conformations. We show how the requirements can be specified as CSP processes and how CSP process refinement can be used for verifying requirement conformation.
- A framework for dynamic analysis of the effect on safety conservation of atypical human task execution behavior.
  - We provide methodology for monitoring a the execution of a computer system being guided by a human operator based on the JavaMOP [30] monitoring framework.
- Framework for modeling and analyzing protected human operator behavior. Protected behaviors may contain both the recommended behavior of executing a task and atypical and possibly erroneous task execution behavior. Protected behaviors collectively form a protection envelope of *tolerable* human operator task execution behavior.
  - We present the logic and format suitable for specifying protection envelope and safety properties: invariants specified in Linear Temporal Logic (LTL).
  - We present methodology for obtaining a recommended task specification abiding model from an unrestrained behavior model. The unrestrained behavior model is specified using Concurrent Game Structures. The recommended task specification restricts the unrestrained behavior model to provide a recommended behavior model.

    - We present methodology for obtaining a protection envelope abiding model from an unrestrained behavior model. Given an unrestricted behavior CGS model and a protection envelope property, we refine the unrestricted human operator behavior in accordance with the protection envelope property.
    - We prove the maximality of the protection envelope abiding model.
    - Methodology for analyzing the boundaries of the safe tolerable behaviors by using different protection envelopes.
    - We present our prototype tool, called “Tutela”, for assisting in executing the framework.

- Methodology to combine notions from runtime monitoring and protection envelope to ascertain the human operator responsible for safety requirement violation.

- Framework for modeling and analyzing the robustness of protected human operator behavior:

    - Given a protection envelope property, we assess how robust it is:
        * We present a method for creating a complete unrestrained behavior model from a sparsely encoded unrestrained human behavior model encoded in a well established verification modeling language.
        * We present how to determine whether the common variations of human recommended task execution behavior are protected or not.
        * We show how to suggest a protection envelope in the absence of a protection envelope.
        * We present an extension to our prototype tool Tutela that implements this framework.

- Enhancements of the Concurrent Game Structures (CGS) for more suitable modeling of human task execution behaviors.

    - *i*ncomplete information *N*ondeterministic Concurrent Game Structure (iNCGS) is an extension of Concurrent Game Structures which supports *n*ondeterministic transition system and has capability of expressing *i*ncomplete information about the human operators. We provide an Alternating Time Temporal Logic (ATL) model checking algorithm for iNCGS.
    - We present nondeterministic strategies for players in a CGS. We show how nondeterministic strategies can be a better model for human behaviors.
    - We provide *strategic semantics* of ATL and ATL* using iNCGS. Strategic semantics provides a more succinct and accurate method of stating whether a specific human behavior can model a guarantee specified using ATL.

1.2 Organization

We have organized the rest of the dissertation as follows: In Chapter 2 we present our initial attempt in modeling and analyzing human operator task execution behaviors. We present our framework for dynamic analysis via monitoring of human operator behaviors in Chapter 3. In Chapter 4, we present our framework for modeling and verifying protection envelopes. Here we also present how to identify the human operator responsible in case of safety violations using runtime monitoring and protection envelopes. In Chapter 5, we present our framework for studying the robustness of a protection envelope. In Chapter 6, we present a nondeterministic CGS model capable of encoding the notion of information hiding to provide a flavor of modularity to the modeled human operators and system components. In Chapter 7, we survey existing works related to different parts of the dissertation.

Chapter 2

Formalizing Recommended Task Specifications

Task analysis is the study of what physical actions and cognitive thought processes a user needs to perform to achieve a task objective. Task analysis determines the steps necessary for a single person or a group of people to execute a task. It also involves considering the time required to perform the tasks, task allocation, environmental effect on subtask selection and how activities belonging to a task fit with each other. The outcome of task analysis is usually a hierarchical decomposition of the given task into simpler subtasks. The analysis provides an understanding of how the subtasks fit together. We focus only on the manner of execution of the steps in a task description demonstrated by a human operator while operating a computer system. Every day human operators control the course of execution of many computer systems in diverse sectors of life. These human operators are usually given a task specification to be followed to operate the computer systems. The task specification that is provided to human operators is assumed to be optimal and safe from the point of view of the goal of the computer system. This task specification is usually recommended by the system designers to be followed by the operators. The recommended task specification is thus a human generated guide for operator behavior. Just as operator deviations from a recommended task specification can be disastrous for a computer system, a wrongly specified recommended task specification itself can be hazardous for the system. In this chapter we present how we can model and verify recommended human operator tasks using process algebra.

2.0.1 Organization

We present an example scenario in Section 2.1. In Section 2.2 we present how to model human task execution sequences using process algebra. In Section 2.3 we present how to specify system requirements using process algebra. In Section 3.1 we present reasoning behind preferring temporal logic formulation of system requirements over process algebraic modeling of system requirements.

2.1 Example Scenario: Automated Identification and Data Capture

Correct identification and correlation of data and patients is very important in health care. Everyone has heard horror stories about patients who got an operation intended for another patient [29], died because they got the wrong medication [68], or had surgery that incorrectly identifies a limb that needed to be amputated. Samples must be very carefully handled in labs and there is an especially important gap between the collection of a reading and its entry into an Electronic Health Record (EHR). As a result, health care facilities such as hospitals, labs, and clinics have rigorous task specifications for clinical personnel that involve careful identification. Clinicians, patients and samples are all labeled with badges, arm tags and stickers respectively for use in many procedures.

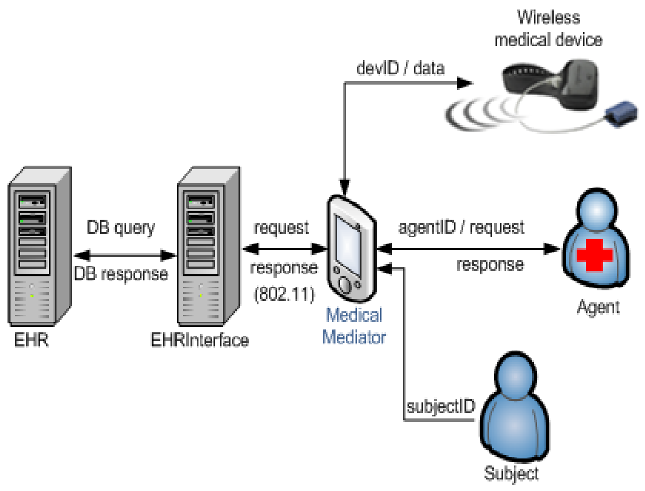
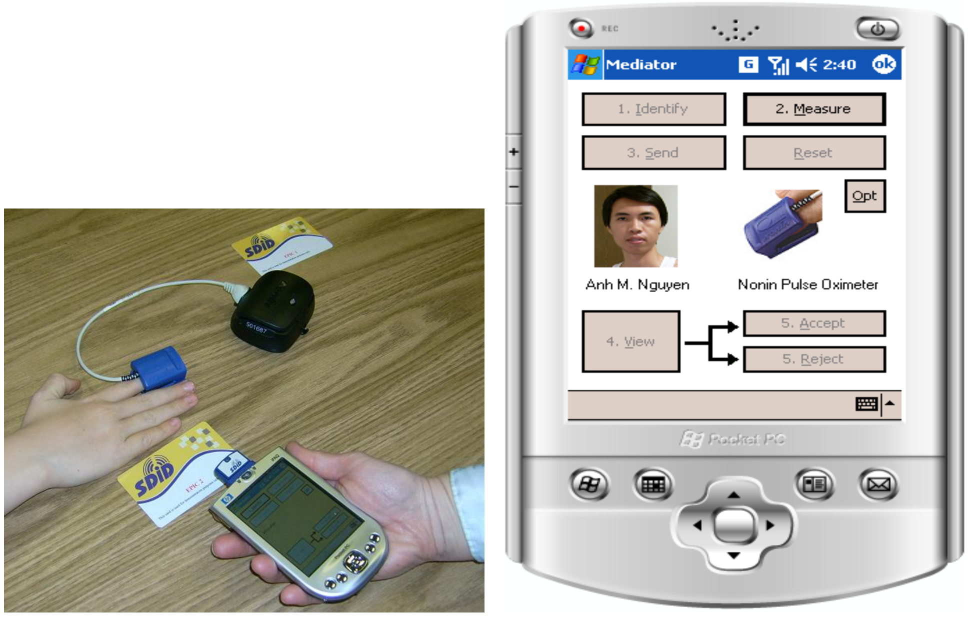


Figure 2.1: AIDC system design

Hospitals are making efforts to improve medical data collection process by using *A*utomated *I*dentification and *D*ata *C*apture (AIDC). Traditionally hospital personnel follow a comparatively manual procedure for this task. For instance, a nurse collects the temperature of a patient on a mechanical scale and writes it on into a paper file. With the emergence of EHRs, the nurse then enters this information into an EHR at a later stage. But if wireless medical devices were used, the task could have been streamlined. For instance, a wireless scale can have a Blue-tooth link to a computer. A reading from the scale can be transferred directly to the computer with the nurse providing other necessary identifications. *Or* the scale can connect to a Personal Digital Assistant (PDA) carried by the nurse by Blue-tooth. The PDA can display the reading and enter it via a WiFi link into the EHR. A project involving building a prototype system for AIDC, modeling and verifying it was undertaken at our university. The prototype system software was developed by Anh Nguyen. We performed the task of formally modeling and verifying the prototype. This prototype



(a) Mobile Mediator (b) Mobile Mediator GUI

Figure 2.2: AIDC system with mobile mediator

AIDC system will be used as the example scenario in several different parts of this dissertation. In the case of the prototype system built at our university, a pulse oximeter connects by Blue-tooth to a PDA (which we will subsequently refer to as a *medical mediator*), which uses an RFID reader to identify the patient and the clinician before entering the results into an EHR via a WiFi link. We added the additional step of identifying the device taking the reading to obtain the following nurse task description:

i. Identify the patient by scanning the patient's identification tag.

ii. Identify the device to be used to take a reading by scanning its identification tag.

iii. View the information to verify whether the correct entities were identified.

iv. Take a reading

v. Check the reading to see if it is acceptable for entry into the EHR and

vi. Approve the reading if it is.

This task description can be augmented by some simple recovery instructions like using a reset function if the entities are not correctly identified or the reading is not satisfactory. The goal of the overall system is to ensure that there is correct association of patient and data in the EHR.

2.2 Formal Model for Task Specification

The task specification for a human operator needs to be encoded using a formalism that is expressive enough to encode the sequence of subtasks that needs to be performed by the human operator. During the process of executing the task, the human operators will interact with various components of the computer system. Thus the formalism should also be capable of capturing the interactions between the human operators and the computer system components. The formalism should also be capable of modeling of the change in the state of the system and human operators as they interact with each other and perform tasks. The formalism should enable expressing the safety requirements of the computer system that the system must conform to irrespective of the behavior of the human operators. We chose a process algebraic approach towards modeling human task execution behavior.

2.2.1 Process Calculus Approach

We first demonstrate the use of *Communicating Sequential Processes (CSP)* to formally model human operators and computer systems [51]. CSP is a process algebraic notation for concurrency and communication. A simplified grammar of CSP is given below. It is based on a primitive set of events e and booleans b.

$$P, Q ::= \mathsf{Stop} \mid \mathsf{Skip} \mid \mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid e \rightarrow P \mid P;Q \mid P \,|||\, Q \mid P[\![\{\mathsf{E}\}]\!]Q \mid P \,\Box\, Q \mid P \sqcap Q \mid P \setminus \mathsf{E}$$

The constant Skip indicates successful completion while Stop indicates a deadlocked process.

The process if b then P else Q behaves like P if b holds and like Q otherwise.

Prefixing The process $e \rightarrow P$ first does e and then behaves like P.

Sequencing The process $P;Q$ behaves like P until P terminates, and then behaves like Q.

Interleaving The process $P \,|||\, Q$ denotes arbitrarily interleaved parallel asynchronous composition of the two processes P and Q.

Synchronization The process $P[\![\{\mathsf{E}\}]\!]Q$ models parallel synchronized execution of two processes P and Q where their synchronization is defined with respect to the occurrence of an event from the interface event set E. An event $e \in \mathsf{E}$ can only occur if each component process can execute e.

External Decision The process $P \,\Box\, Q$ behave either like P or Q, but the choice between them is made externally by the environment.

Internal Decision The process $P \sqcap Q$ behave either like P or Q, but the choice between them is made internally by the environment.

Abstraction The abstracted process $P \setminus \mathsf{E}$ provides a mechanism for hiding events. Each event e in the set E is rendered unobservable by this process definition.

The operators $|||$, $[\![\mathsf{E}]\!]$, $\square$, and $\sqcap$ are associative and commutative. We will abbreviate $P(x_1) \square \ldots \square P(x_n)$ by $\underset{x \in X}{\square} P(x)$ where $X = \{x_1, \ldots, x_n\}$, and similarly for $|||$ and $\sqcap$. All sets will be assumed to be finite. We will further abbreviate $\underset{\{C.x | x \in X\}}{\square} C.x \to P(x)$, where X is an understood set of values for a communication, by $C?x \to P(x)$ meaning the possible receipt of a value, and write $C!v \to P$ for $C.v \to P$, representing the sending of a value v. We will refer to the modifier C in a set of events $\{C.x | x \in X\}$ as a communication channel, or channel for short.

2.2.2 Human Behavior Model using CSP

Any process calculus is a language for describing the communication, interaction and synchronization among different entities where each entity is essentially a process. We can consider a human operator as a CSP process capable of interacting, communicating and operating synchronously with a computer system. CSP provides a uniform methodology for modeling both the human operators and the system components. The steps performed by a computer system can be considered as CSP processes. Thus the parallel synchronous composition of CSP processes corresponding to human operators and computer systems can be an abstract model to be analyzed and verified. CSP also allows for modeling of infinite state systems. We will present the advantages of using CSP to model human behavior as we present our CSP modeling of the AIDC scenario.

2.2.3 AIDC Scenario using CSP

Our approach to modeling AIDC is to represent each of the environment elements (entities supplying identity and information: RFID tags as shown in Figure 2.1, the patient), the human task specification, the identification system and the identification platform (the mobile mediator, the back-end database: EHR) as CSP processes to be composed with synchronization. The environment and the identification platform should be thought of as fixed and the purpose of their specification is to capture existing functionality. The specifications for each of the human task specification and the identification system should be thought of as stating the required behavior to be implemented. The composition of the human task specification and the identification system is the total "system" to be supplied. However, since the human behavior can not be guaranteed, it is critical that its specification be broken out separately to clarify the precise limitations of the system deployed.

For each type of pairing of entities in the environment, human agents, identification system components and platform components, we associate a parameterized family of channels, where the parameters give the specific implicit and explicit participating entities. In our examples, we give the parameters as indexing subscripts and superscripts. To the extent one can say one party "owns" a channel, we use the subscript to indicate the owner. The superscripts indicate information about the other participant(s) in the communication, and the source(s) of the data being communicated. These are mathematical labeling of the channels by "real world" entities and are not visible to the system being implemented; only the data transmitted is actually visible. However, these indices play a critical role in enabling us to state the needed correspondence properties guaranteeing that the recorded data had originated from the expected source, *i.e.*, they have the expected identity.

We explore a scenario at a hospital where nurses take patient readings from devices capable of reporting results via wireless. The challenge here is to ensure that the correct reading gets associated with the correct patient in the hospital database. To aid in ensuring this, all elements of the hospital environment are equipped with identification tags, and the nurse is required to identify with those tags the patient and the device being used, and then the system should ensure that the data collected are associated with the identified patient and device. The whole process can be seen as an interaction among the physical actions taking place in the environment, like the nurse taking the reading using a device and the actions in the electronic system, which receives, interprets and stores the readings collected from patients.

In accordance with the general methodology outlined above, the environment comprises the patients, the medical devices, and all tags associated with these entities. The nurse must adhere to the human task specification. The medical mediator (the PDA plus readers and software) is the identification system, and the combination of the EHR and its interface make up the identification platform. To model the RFID tags, the patients, the nurses, the mobile mediators, the medical devices, the EHR interface and the back end EHR we will use the CSP processes Tag, Pat, Nurse, Med, Device, $\mathsf{EHRInterface}$ and EHR respectively. Some of these processes will be further described in terms of subprocesses, such as $\mathsf{TakeCkReading}$ for Nurse. The channels are as follows: HCI^n_m is used for interactions between the nurse n and the medical mediator m; $\mathsf{RFIDChan}^{n,m}_o$ for communicating RFID tag numbers for a tagged object o to a medical mediator m operated by nurse n; $\mathsf{BTAddr}^{n,m}_d$ is the Blue-tooth connection between m and device d selected by n, while BTScan_m represents scanning the radio frequencies for blue-tooth devices; EHRCh^m is the channel for m to communicate with the EHR interface to get names associated with id numbers, and $\mathsf{EHRBECh}^m$ is the channel for m to use to store information in the EHR.

Human task specification. In Section 2.1, we introduced the nurse's task specification. We now present CSP process for the nurse.

$$
\begin{array}{l}
\mathsf{Nurse}(n,m) = \\
\quad \mathsf{HCI}^n_m!\mathsf{GetID} \rightarrow \bigsqcap_p (\mathsf{HCI}^n_m!(\mathsf{RFIDChan}^{n,m}_p) \rightarrow \\
\quad \bigsqcap_d (\mathsf{HCI}^n_m!(\mathsf{RFIDChan}^{n,m}_d) \rightarrow \mathsf{HCI}^n_m?x \rightarrow \\
\quad \mathsf{if}\, x = (\mathsf{Name}(p), \mathsf{Name}(d)) \\
\quad \mathsf{then}\, (\mathsf{HCI}^n_m!\mathsf{Yes} \rightarrow \mathsf{TakeCkReading}(n,m,p,d)) \\
\quad \mathsf{else}\ \ \mathsf{if}\ \ x = \mathsf{Error}\ \ \mathsf{then}\, (\mathsf{HCI}^n_m!\mathsf{OK} \rightarrow \mathsf{Skip}) \\
\qquad \mathsf{else}\, (\mathsf{HCI}^n_m!\mathsf{No} \rightarrow \mathsf{Skip})))
\end{array}
$$

The specification relies upon the specification TakeCkReading of the task specification to be followed when taking and verifying a reading for a patient. Before presenting the CSP process for that phase, we provide explanation of the CSP encoding presented above.

- The nurse first presses a GetID button on the display of the mobile mediator. This action is encoded as a communication between the nurse and the mobile mediator $\mathsf{HCI}^n_m!\mathsf{GetID}$. All communication between a particular nurse n and a mobile mediator m is performed via sending and receiving of messages over the human computer interface channel indexed by n and m: HCI^n_m. The nurse manifests her intention of scanning RFID tags by sending the GetID message.

- Then the nurse indicates the patient whose tag she wants to scan by pointing the mobile mediator to that patient's tag. The patient that the nurse wants to indicate is chosen by the nurse herself. Thus the choice of patient is an internal decision for her. We model this part by $\bigsqcap_p (\mathsf{HCI}^n_m!(\mathsf{RFIDChan}^{n,m}_p))$. Whichever patient the nurse wants to handle is indicated by the sending of the communication channel between that patient's RFID tag and the mobile mediator by the nurse to the mobile mediator. Thus this is again a communication over the channel HCI^n_m where the value that is communicated is another channel: $\mathsf{RFIDChan}^{n,m}_p$. It indicates the channel for communication between the RFID tag of the patient p and the mobile mediator m associated with the nurse n. Similarly the nurse indicates the device that should be scanned by sending the message $\mathsf{RFIDChan}^{n,m}_d$ over the same channel HCI^n_m.

- Now the nurse needs to read the information about the scanned entities on the mobile mediator display. This is modeled by the nurse waiting to receive message over HCI^n_m.

- After reading the displayed information, the nurse needs to decide if the displayed information is associated with the patient and device she had intended to use. If they are, then she affirms the validity of the displayed information: $\mathsf{HCI}^n_m!\mathsf{Yes}$ and tries to collect a reading in the next step. Otherwise, there are two possibilities. First: it may be the case that she decides that the information displayed does not correlate with the patient and the device she intended to use and she indicates this by sending out $\mathsf{HCI}^n_m!\mathsf{No}$. Second: suppose the back end database was not able to successfully find information about the scanned entities. Then an error will be indicated to the nurse via the mobile mediator. In case of such an error message the nurse indicates to the mobile mediator that she understands there is a problem with the scanned identities and sends out an OK to the mobile mediator. In real life, the nurse will most probably start over the entire AIDC procedure.

We now move on to the reading collection phase. When the nurse is ready to take a reading she pushes a GetReading button on the medical mediator, then selects a blue-tooth device from the list displayed assuming it is present, examines the results of the reading and decides whether to store the results if they are meaningful. In CSP, we can encode this as:

$$
\begin{array}{l}
\mathsf{TakeCkReading}(n, m, p, d) = \\
\quad \mathsf{HCI}^n_m!\mathsf{GetReading} \rightarrow \mathsf{HCI}^n_m?X \rightarrow \\
\quad \mathsf{if}\, \mathsf{BTAddr}^{n,m}_d \notin X \\
\quad \mathsf{then}\ \mathsf{HCI}^n_m!\mathsf{No} \rightarrow \mathsf{Skip} \\
\quad \mathsf{else}\ \mathsf{HCI}^n_m!\mathsf{BTAddr}^{n,m}_d \rightarrow \mathsf{HCI}^n_m?data \rightarrow \\
\quad \mathsf{if}\ data = \mathsf{Error} \\
\quad \mathsf{then}\ \mathsf{HCI}^n_m!\mathsf{OK} \rightarrow \mathsf{TakeCkReading}(n, m, p, d) \\
\quad \mathsf{else}\, ((\mathsf{HCI}^n_m!\mathsf{Yes} \rightarrow \mathsf{Skip}) \sqcap \\
\qquad\quad (\mathsf{HCI}^n_m!\mathsf{No} \rightarrow \mathsf{Skip}))
\end{array}
$$

2.2.4 Advantages of using CSP for Modeling Humans

We now present the advantages of using CSP as the formalism for modeling human behavior.

1. Communication: CSP has a simple and elegant way of modeling communication. One can very easily define channels and send and receive messages over them. For example, let us consider the very first part of the nurse process: $\mathsf{HCI}^n_m!\mathsf{GetID}$. Here, we define the channel HCI^n_m for communication between the nurse n and the mobile mediator m. A hospital has many nurses and can have many mobile mediators. CSP provides a simple method of defining channels of communication for each pairings of a nurse and a mobile mediator through such indexed channels. Thus we can unambiguously indicate the channel of communication for the exact nurse and mobile mediator. Moreover, CSP does not impose any restrictions on the type of message that can be defined or more importantly sent over a channel. We sent abstract messages like GetID, Yes and we also sent channels $\mathsf{RFIDChan}^{n,m}_p$.

2. Correlation of identities: Correspondence of identities of objects at progressive stages of a CSP process can be very easily achieved by using indexes for the channels and variables. For example when we wanted to describe that a particular patient is internally chosen by the

nurse, we indicated that choice of a patient by p. If we examine the CSP code for the RFID tag scanning phase, we see that we can use the same p as an index for the RFID channel for the patient, for the decision part of the nurse, for indicating the same patient to the reading collector process $\mathsf{TakeCkReading}(n, m, p, d)$.

3. Internal nondeterministic decision: A nurse is assigned several patients each day. She decides which patient she is going to handle first. This decision is taken internally by her and we model her decision procedure by the outcome of an internal nondeterministic choice by the CSP process for the nurse: $\underset{p}{\sqcap}(...)$. Let us consider a more involved example to illustrate the usefulness of internal nondeterminism operator. The nurse process presented in this section corresponds to the recommended behavior for the nurse. If the displayed information is associated correctly with the patient and the device she had intended to use then she affirms their validity. Only if there is a problem with the displayed information, she sends out No. However, for a nurse not following this suggested course of action may send out a No even if the displayed information is correct. Even worse, she may send out a Yes even if the displayed information is not associated with the patient and the device she was handling. Thus the decision procedure is totally internal to the nurse and the outcome is nondeterministic. CSP allows such internal decisions to be modeled easily by the $\sqcap$ operator. A possible model for a nurse can be:

   ```
   $\mathsf{Nurse}(n, m) =$
       ...
       if $x = \mathsf{Error}$ then $(\mathsf{HCI}_m^n!\mathsf{OK} \to \mathsf{Skip})$
           else $((\mathsf{HCI}_m^n!\mathsf{Yes} \to \mathsf{TakeCkReading}(n, m, p, d))\sqcap(\mathsf{HCI}_m^n!\mathsf{No} \to \mathsf{Skip}))$
   ```

   Here if the mobile mediator displays that information about the patient and the device could not be retrieved from the database, then the nurse takes the recommended action. However, in the case where some information was retrieved from the database about the patient and the device, the nurse makes a nondeterministic internal decision about whether to validate the displayed information or not.

We will present some other advantages of CSP as we present more CSP encodings.

Environment. The tags in the environment should be modeled as processes that repeatedly announce their identification number to any party listening. All we directly know about a patient is her tag. A blue-tooth device is similar to a tag, except that it chooses a fresh set of data for each communication, instead of a single identification number. A device is viewed as a combination of a tag and a blue-tooth device. Separate from the individual blue-tooth devices, we will have a channel that is used for detecting all available blue-tooth devices. These are specified by:

$$\begin{aligned}
&\mathsf{Tag}(o) = \underset{n,m}{\Box}\mathsf{RFIDChan}_o^{n,m}!(\mathsf{IDnum}(o)) \to \mathsf{Tag}(o)\\
&\mathsf{Pat}(p) = \mathsf{Tag}(p)\\
&\mathsf{BTDev}(d) = \underset{n,m}{\Box}(\underset{data}{\sqcap}(\mathsf{BTAddr}_d^{n,m}!data \to \mathsf{BTDev}(d)))\\
&\mathsf{Device}(d) = \mathsf{Tag}(d) \,|||\, \mathsf{BTDev}(d)\\
&\mathsf{BTDevs} = \underset{m}{\Box}(\underset{X \subseteq \mathsf{BTDevices}}{\sqcap}\mathsf{BTScan}_m!X \to \mathsf{BTDevs})
\end{aligned}$$

Identification Platform. The two database components that are relied upon by the medical mediator are the EHR and the EHR interface. For the interface, we assume that it can somehow determine if the tag identifiers make sense, sort them and accompany them with the corresponding names. The EHR is just required to accept whatever it is sent.

$$\begin{aligned}
&\mathsf{EHRInterface}(m) = \mathsf{EHRCh}^m?x \to\\
&\quad((\mathsf{EHRCh}^m!\mathsf{Error} \to \mathsf{Skip})\sqcap\\
&\quad(\sqcap_{\left\{\begin{array}{l}(n_1,n_2)|\exists y_1\, y_2.(y_1,y_2) = x \vee (y_2,y_1) = x\\ \quad\wedge(n_1 = \mathsf{Name}(y_1) \wedge n_2\mathsf{Name}(y_2))\end{array}\right\}}\\
&\quad\mathsf{EHRCh}^m!n_1 \to \mathsf{EHRCh}^m!n_2 \to\\
&\quad\mathsf{EHRCh}^m?r \to \mathsf{EHRCh}^m!FormatRaw(r) \to\\
&\quad\mathsf{EHRCh}^m?z \to \text{ if } z = \mathsf{Auth}(n,m)\\
&\quad\text{then } (\mathsf{EHRBECh}^m!(n,m,y_1,y_2,EHR(r)) \to \mathsf{Skip})\\
&\quad\text{else } \mathsf{Skip}))\\
&\mathsf{EHR} = \underset{m}{\Box}(\mathsf{EHRBECh}^m?x \to \mathsf{EHR})
\end{aligned}$$

Identification System. The medical mediator is the system component that allows the nurse to communicate with various environmental elements like the RFID tags as well as system components like the EHR. We give the high-level CSP specification here.

$$
\begin{aligned}
&\mathsf{Med}(n,m) = \mathsf{HCI}^n_m?\mathsf{GetID} \rightarrow \mathsf{HCI}^n_m?x_1 \rightarrow x_1?y_1 \rightarrow \\
&\quad \mathsf{HCI}^n_m?x_2 \rightarrow x_2?y_2 \rightarrow \mathsf{EHRCh}^m!(y_1,y_2) \rightarrow \mathsf{EHRCh}^m?n_1 \\
&\quad \rightarrow \text{ if } n_1 = \mathsf{Error} \text{ then } \mathsf{HCI}^n_m!n_1 \rightarrow \mathsf{HCI}^n_m?\mathsf{OK} \rightarrow \mathsf{Skip} \\
&\quad \text{else } \mathsf{EHRCh}^m?n_2 \rightarrow \mathsf{HCI}^n_m!(n_1,n_2) \rightarrow \mathsf{HCI}^n_m?z \rightarrow \\
&\quad \text{if } z = \mathsf{Yes} \text{ then } \mathsf{MedRead}(n,m) \text{ else } \mathsf{Skip} \\
&\mathsf{MedRead}(n,m) = \mathsf{HCI}^n_m?\mathsf{GetReading} \rightarrow \\
&\quad \mathsf{BTScan}_m?X \rightarrow \mathsf{HCI}^n_m!X \rightarrow \mathsf{HCI}^n_m?y \rightarrow \\
&\quad \text{if } y = \mathsf{No} \text{ then } \mathsf{Skip} \\
&\quad \text{else } y?rdata \rightarrow \mathsf{EHRCh}^m!rdata \rightarrow \mathsf{EHRCh}^m?fdata \rightarrow \\
&\quad \mathsf{HCI}^n_m!fdata \rightarrow \text{ if } fdata = \mathsf{Error} \\
&\quad \text{then } \mathsf{HCI}^n_m?\mathsf{OK} \rightarrow \mathsf{MedRead}(n,m) \\
&\quad \text{else } \mathsf{HCI}^n_m?z \rightarrow \text{ if } z = \mathsf{Yes} \\
&\quad \text{then } \mathsf{EHRCh}^m!\mathsf{Auth}(n,m) \rightarrow \mathsf{Skip} \\
&\quad \text{else } \mathsf{Skip}
\end{aligned}
$$

2.3 Safety Properties

There are a variety of different semantics and models for CSP processes. We concentrate on the trace model and the failure model in this work. In a trace model a process is represented by the sequences of communications or events it can perform. In the stable failures model, a process is represented by its failures. A failure for a process, P, is given by a pair (s, X), where s is a trace of P and X is a set of things it can refuse to perform after performing s. We provide some desired safety and liveness properties for the hospital AIDC system. We also provide the specifications for a CSP process by providing the most nondeterministic process satisfying the specification. The CSP process for a hospital AIDC system satisfies a specification S if the CSP process P representing the specification is refined by the AIDC system process. We consider two different types of refinement: trace refinement for safety properties and failure refinement for liveness properties [99]. A process Q is a trace refinement of another process P, denoted by $P \sqsubseteq_T Q$ if $\mathsf{traces}(Q) \subseteq \mathsf{traces}(P)$. A process Q is a failure refinement of another process P, denoted by $P \sqsubseteq_F Q$ if $\mathsf{failures}(Q) \subseteq_F \mathsf{failures}(P)$. If we have a process P and its safety specification S and we have $S \sqsubseteq_T P$, then we can say P is a safe process, as all its traces are traces of the specification, so no bad event can be performed by P. Similarly, failure refinement can be used to express liveness properties.

We now express the environment of a hospital AIDC system as the collection of all the identification tags representing the devices and patients and the devices sending out the readings for patients, together with the electronic records system. These are the "given" components, which we cannot control.

$$\mathsf{Given} = \ (\mathop{|||}_{d \in \mathsf{Dev}} \mathsf{Dev}(d)) \ ||| \ (\mathop{|||}_{p \in \mathsf{Pat}} \mathsf{Pat}(p)) \ ||| \ \mathsf{BTDevs} \ ||| \ \mathsf{EHR}$$

We shall refer to the combination of all instances of nurses and medical mediators as the system. It is important that only one nurse have use a medical mediator at a time. This is reflected in our specification by allowing each medical mediator to "choose" a nurse.

$$\begin{aligned}
\mathsf{System} = \ & ((\mathop{|||}_{m} (\mathop{\sqcap}_{n}(\mathsf{Nurse}(n,m) \ [\![\{y|\exists x.y = \mathsf{HCI}^{n}_{m}.x\}]\!] \\
& \mathsf{Med}(n,m)) \ [\![\{y|\exists x.y = \mathsf{EHRCh}_{m}.x\}]\!] \\
& \mathsf{EHRInterface})) \setminus \{y|\forall n \ m \ p \ d \ x. \\
& y \neq \mathsf{RFIDChan}^{n}_{p}.x \wedge y \neq \mathsf{EHRBECh}^{m}.x \wedge \\
& y \neq \mathsf{BTAddr}^{n,m}_{d}.x \ \wedge y \neq \mathsf{BTScan}_{m}.x\}); \\
& \mathsf{System}
\end{aligned}$$

In the system, we hide away almost all internal and inter-element communication. The only events that we are interested in are the actual scanning of patient and device identifier tags and the ultimate sending of the medical measurement to the EHR for storage after it has been verified by the nurse. This is another aspect of CSP that was very useful. The hiding operator of CSP allowed us to easily abstract away the information we did not want to be observable. The other interesting CSP operator in use here is the synchronization operator. We defined the nurse process and the mobile mediator process separately and here we defined the combined process where they need to synchronize on the communications occurring over the channel between them HCI^{n}_{m}. Thus the mobile mediator can only progress after the nurse has sent the GetID message, the nurse can only verify collected reading after the mobile mediator has displayed them to her.

We now provide the safety specification for the hospital AIDC system, where the specification ensures that if a tuple of data is entered into a database, stating that a particular device, d was used to take a reading x from a patient p by a nurse n, then it had been the case that the nurse n *did* identify p and d and *had* taken a medical measurement x from the patient p using d after identifying them.

$$\begin{aligned}
& \mathsf{Safety} = \\
& \mathop{|||}_{m} (\mathop{\sqcap}_{n}\mathop{\sqcap}_{p}\mathop{\sqcap}_{d}(((\mathsf{RFIDChan}^{n,m}_{p}.\mathsf{IDnum}(p))^{+} \rightarrow \ (\mathsf{RFIDChan}^{n,m}_{d}.\mathsf{IDnum}(d))^{+}) \\
& \sqcap((\mathsf{RFIDChan}^{n,m}_{d}.\mathsf{IDnum}(d))^{+} \rightarrow \ (\mathsf{RFIDChan}^{n,m}_{p}.\mathsf{IDnum}(p))^{+}) \\
& \rightarrow (\mathsf{Skip} \sqcap (\mathsf{BTScan}_{m}?X \rightarrow (\mathsf{Skip} \sqcap \\
& \quad (\text{ if } \ \mathsf{BTAddr}^{n,m}_{d} \in X \text{ then } (\mathsf{BTAddr}^{n,m}_{d}?r \rightarrow (\mathsf{Skip} \sqcap \\
& \quad \mathsf{EHRBECh}^{m}.(n, m, \mathsf{IDnum}(p), \mathsf{IDnum}(d), \mathit{EHR}(r)) \\
& \quad \rightarrow \mathsf{Skip})) \text{ else } \mathsf{Skip})))))); \mathsf{Safety}
\end{aligned}$$

Now, we are ready to specify the desired safety and liveness properties. Let $\mathsf{Vis} = \{y.(\exists n \ d \ x. \ y =$

$\mathsf{RFIDChan}_d^{n,m}.x) \vee (\exists m\ z.\ y = \mathsf{EHRBECh}^m.z)\}$. The properties then are as follows:

$$\mathsf{Safety}[\![\{\mathsf{Vis}\}]\!]\mathsf{Given} \sqsubseteq_T \mathsf{System}[\![\{\mathsf{Vis}\}]\!]\mathsf{Given}$$
$$\mathsf{LIVE} \sqsubseteq_F \mathsf{System}[\![\{\mathsf{Vis}\}]\!]\mathsf{Given}$$

where $\mathsf{LIVE} = \underset{x}{\sqcap} x \rightarrow \mathsf{LIVE}$. LIVE is the most nondeterministic process that is deadlock free. The first property states that no wrong, random, non-associated tuple gets stored in EHR and the second property states that the nurse can eventually take steps and will not get deadlocked.

We have sketched proofs for the above results based on their semantics using traces of events. Due to the large number of cases to be considered, we feel that such a proof is easily subject to accidental omissions and hence is best checked by automated assistant. To this end we have encoded the CSP processes described in this paper two such tools. The first system we used was the fully automated CSP failures-refined prover FDR2 (Failures-Divergence Refinement) [1]. A straightforward encoding led the system to run for hours, probably indicating divergence due to the large number of internal and external choices. The second system we used was the extension CSP-prover [60] to the interactive theorem prover Isabelle [2]. In this system, there is a restricted type system for communications and the encoding entailed creating a number of functions for coercing data into this type system. Also, certain operators were lacking (index interleaving, in particular), and so we had to expand the code to accommodate this lack. As of now the proofs in CSP-prover of the above two results are incomplete. We conclude from this effort that there are interesting challenges to automating proofs of task specifications using CSP for entities such as the ones for the mediator.

2.4 Summary

This chapter has shown how we can formalize human operator task executions using a process algebraic approach, more specifically CSP. CSP provides a high level uniform abstract formalism for modeling the computer systems along with their human operators. We have also presented how system requirements can be specified using CSP. We demonstrate how we can model and analyze the operation of a system being guided by a human operator.

Chapter 3

Runtime Monitoring of Human Operators

In this chapter we present how the combined interactions of a computer system and human operator can be monitored at run time to detect safety requirement violation.

3.1 Introduction

The process algebraic approach to modeling human operators, computer systems through their actions and interactions provides a procedural method for formally specifying them. Computer systems are highly procedural in nature. Human recommended task specification also has a procedural flavor. It comprises of the sequences of actions that need to be performed to ensure successful completion of task. Computer software is usually composed of a collection of software modules. Each module implements a specific procedure of actions to be performed. A computer system is designed to perform a task to achieve a goal. For example, an ATM banking system software is designed to perform tasks of performing the tasks of taking deposits, providing withdrawals and retrieving customer account balance information. The requirement of the system is to maintain integrity of the banking database and protecting client accounts from errors. For example: if a customer performs a transaction then only that customer's account should be impacted. Similarly in the AIDC scenario the computer system is there to help collect and store correct data about the patients. A safety requirement of the AIDC system can be:

Safety if a tuple $\langle n, m, p, d, r \rangle$ gets inserted into the EHR, then it has been the case that the nurse n had identified *the* patient p, had identified *the* device d, and had collected the data r from *patient* p using *the* device d.

As one can observe, the safety condition expressed using CSP in Section 2.3 is as different as possible from this natural description of the safety property. System requirements are usually logical expressions. CSP allows logical reasoning through several refinement relations. These refinement relations are expressed over CSP processes. Hence the only mechanism provided by CSP to express system requirements is through the description of CSP processes conforming to those requirements. This approach can become cumbersome and error-prone in complex scenarios. Safety properties are provided by system designers. Since the system designers are human beings, the given requirements may have errors in them. If they are required to express system requirements using complex CSP processes then it becomes too much of a burden for them. Usually the system designers only provide an English description of the system requirement. The translation of the requirement into a CSP process can introduce errors in the safety requirement itself, thereby nullifying the gains of analyzing against it.

3.1.1 Temporal Logic for Requirement Specification

The system requirements are usually temporal in nature. The safety property presented in Section 3.1 states that for every tuple of information recorded in the EHR, there had to have been a previous point in time when the nurse had identified that patient and so on. An ATM banking software must guarantee that if \$$X$ is withdrawn from the account of customer c then it must be the case that at the immediately preceding point in time the same customer c had requested a withdrawal of \$$X$ from his account. Thus we contend that using some form of temporal logic will be most suitable for specifying system requirements. The most widely used temporal logic is Linear Time Temporal Logic (LTL) [93]. The syntax of an LTL formula φ is as follows: $\varphi = p$ (atomic proposition) $\mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \Diamond\varphi \mid \Box\varphi$. $\bigcirc\varphi$ indicates that in the immediate next step in time the temporal property φ will hold. $\varphi_1 \,\mathcal{U}\, \varphi_2$ indicates that at some point in the future, φ_2 will hold and at each point in time before that φ_1 will continue to hold. $\Diamond\varphi$ denotes that eventually at some point in time the property φ will hold. $\Box\varphi$ denotes that from now on at every step the property φ will hold. There is a past-time extension of LTL called PTLTL [47, 46, 87, 79]. It has the interesting operators $\blacklozenge\varphi \mid \varphi_1 \mathcal{S}\, \varphi_2 \mid \blacksquare\varphi$. Here, $\blacklozenge\varphi$ states that at some previous point in time φ was true. $\varphi_1 \,\mathcal{S}\, \varphi_2$ denotes that φ_2 held at some earlier point in time and since then at every point in time φ_1 held. $\blacksquare\varphi$ indicates that at all earlier points in time φ held.

AIDC Scenario Safety Property in Temporal Logic

The safety property of the AIDC scenario as presented in Section 2.1 using PTLTL is:

$$\begin{aligned}\mathtt{getsStoredInEHR}(\mathtt{n},\mathtt{m},\mathtt{p},\mathtt{d},\mathtt{r}) \quad \rightarrow \quad & \blacklozenge\mathtt{scannedPatient}(\mathtt{p}) \wedge \blacklozenge\mathtt{scannedDevice}(\mathtt{d}) \\ & \wedge \blacklozenge\mathtt{capturedData}(\mathtt{p},\mathtt{d},\mathtt{r})\end{aligned}$$

The concept of a declarative temporal specification of system requirement presents the question of whether we can consider runtime verification of the computer system under consideration while being operated by human operators.

3.1.2 Runtime Verification

Runtime verification or runtime monitoring comprises of a monitor module monitoring an execution of a software module to determine whether the execution trace of the software module conforms to a requirement specification. Unlike static verification where an abstract model of the computer software is verified, runtime monitoring is performed on the actual software program. The contention is that although an abstract model of a system has been proved to be correct, the real implementation may have some unexpected behavior when facing the real environment. These atypical behavior may even be unsafe, thereby reducing the credibility of the verified abstract model. Static analysis verifies each possible execution of the abstract software model. Runtime analysis is usually performed on a specific execution trace. The execution trace may be from an actual or test deployment of the system. The system requirement is usually specified using temporal logic. The desired goal of the monitor system is to be able to detect a requirement violation and perform suitable steps to guide the system to safety.

The concept of runtime monitoring presents this question to us: Is there a way we can monitor a computer system as it gets operated by the human operators to detect violation of the safety of the computer system? The answer is: yes we can. One approach is to design monitors for a computer system and monitor its execution while being operated by humans. Another approach can be to create a model of a human operator actions performed while executing a specific task. We focus on this second approach in this chapter. We can create a monitoring program to monitor the execution of a computer system as it interacts with the human operator action model to detect when a safety requirement is violated and perform necessary steps to remedy the situation. In this chapter we present a methodology for human operator and computer system interaction monitoring to assess the safety of operation of the computer system by the human operator. The focus of this work is not on monitoring and verifying just a computer system but on verifying a computer system whose operation is being controlled by human operators to some degree. We present our framework in Section 3.2. We present a case study in Section 3.3.

3.2 Framework for Monitoring Computer Systems Operated by Human Operators

The execution of every computer system is guided by some human operator to some degree. The human operator uses his past experience, expertise, sense of judgment and current frame of mind to guide the operation of a system. In case of emergencies, many safety critical systems like aircrafts, nuclear power plants rely on the judgment and decision making capability of the human operators to steer the system towards safety. The scenarios presented in Chapter 1 (Three Mile Island Accident) and Sections 6.2 (Japan Airlines Incident) are examples of such situations. Although the system designers rely on the human operators' capability to ensure safe operation of a system, the human operators may get influenced by other environmental elements and their decisions may become erroneous. Even unwillingly they may steer a system towards a hazardous situation instead of guiding it to safety. Hence there is a need to incorporate measures to ensure that human operator introduced errors be captured and dealt with.

The issue that arises now is how to determine that a human operator behavior will not cause unsafe computer system behavior. We have shown that given a recommended human task execution behavior, we can verify that the system remains safe when operated by that recommended behavior. However, the analysis is performed on a model of the system. What if when the actual computer system is deployed, unexpected behavior of the human operators causes even more unexpected result from the system? What if the fault actually lies with the computer system? A framework that will allow us to determine whether a human operator behavior is safe for an actual system will help us answer these questions. Since human operators are so unpredictable in nature there can be numerous possible behaviors. It is not feasible to study each and every behavior of a human operator exhibited while interacting with a system. Hence we suggest off-line monitoring of a software system with a specific human behavior to determine whether the system can maintain safety when facing that behavior.

3.2.1 The Monitoring Framework

The idea behind the monitoring methodology is very simple as presented in Figure 3.1. The methodology assumes existence of a monitoring framework that can be used to create the monitor specification and execute it. The parts of a simple framework for monitoring human behavior is presented below:

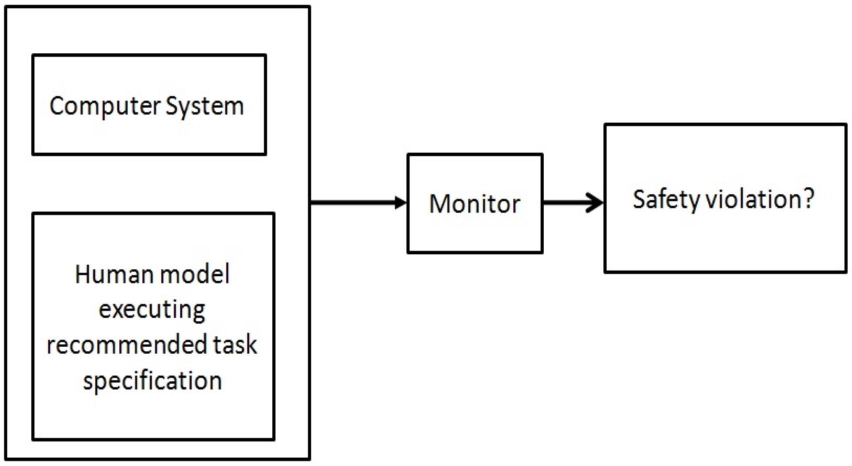


Figure 3.1: Monitoring a system being operated by a human

- Model a human operator behavior in a language acceptable to a monitoring tool.
- Identify the safety requirement of a system. Determine the procedure to be followed if a violation to the safety requirement occurs.
- Create a monitoring specification for the combination of the human operator and the system.
- Execute the human behavior model, the computer system, and the monitor to ascertain if any security requirement violation occurs.

The human behavior can be considered to consist of the steps or actions he takes in different states. This can be thought of as his game plan in operating a computer system. And also this can be thought of as his execution of his task specification. A task specification prescribes the action sequences an operator should execute. The task specification or human behavior will then cause a trace of the system to be created at runtime. This execution trace of the computer system is guided by the human operator. If this execution trace does not lead to any requirement violation then we can state that the corresponding human behavior is a safe one. Suppose the execution trace created at runtime under the influence of the human operator leads to violation. Then if we assume that the system is safe, then the human behavior is responsible for forcing the system to enter an unsafe state. Later on in Section 4.3, we show how to determine whether the system itself or the human operator is responsible for the safety violation.

3.2.2 Framework with Chosen Monitoring Software

The monitoring software chosen by us is JavaMOP [30] developed in FSL (`fsl.cs.uiuc.edu`) at the University of Illinois. JavaMOP is a framework for runtime analysis of a software via monitoring for conformance of a requirement specification. The requirement can be specified using many different logics. We chose LTL and PTLTL (*p*ast time LTL) [47, 46, 87, 79] for the requirement specifications. The monitor recognizes AspectJ [123, 3] events. AspectJ events can be the call to a Java [4] method with or without a specific set of input parameters, exiting from a Java method with or without a specific return value and so on. Each event is considered as an atomic proposition. The LTL or PTLTL property is specified using the event propositions. Given our choice of the monitoring software the framework now becomes:

- Model a human operator behavior in a language acceptable to a monitoring tool.
  - Identify the human operator vocabulary.
  - Encode each human action with Java modules. The human actions will entail calls to the modules in the actual software.
  - Construct a human operator behavior model by composing human action methods. This model will be contained in a Java module.

- Identify the safety requirement of a system. Determine the procedure to be followed if a violation of the safety requirement occurs.
  - Create aspects corresponding to interesting events taking place in the execution of the software program. A system needs to react to the steps that a human operator takes to execute each step in a task description. The interesting events are essentially all the human operator actions or the system reactions to human operator actions. Important system method calls should also get events.
    - For simply a call to a software module, we can create *jointpoints*.
    - For a collection of calls to a specific software module, create a *pointcut*.
    - For including some action that needs to be undertaken before or after a particular jointpoint occurs declare *advices* accordingly.
- Obtain the safety requirements of the system.
  - Considering the set of events defined in the last state as the set of propositions, formulate a PTLTL or a LTL property.
  - Include the appropriate steps that need to be taken in case of a violation of the formulated property.
- Create a monitoring specification for the combination of the human operator and the system.
  - Combine the event set and the property along with violation advice to create the actual monitor program.
- Execute the human behavior model, the computer system, and the monitor in the JavaMOP framework to ascertain if any security requirement violation occurs.
  - Execute the monitor on the execution of the combination of the human operator module and the computer system.

Execution of a monitor on the combination of a human operator behavior and a computer system will present a violation if any. Then the action advice for the requirement violation will take place to hopefully guide the system back to safety.

3.3 Human Task Execution Monitoring Case Study

We will consider the AIDC scenario for our case study. We must mention that we had to use a model Java encoding of the scenario for the case study instead of the actual prototype software that had been built. This scenario is simple enough in that the only active human operator is the nurse. The most noticeable system component is the mobile mediator. The safety of the AIDC procedure is the same from the point of view of the patient and the hospital:

- if the EHR has been asked to store a tuple containing a reading *r* that has been captured using a device *d* from a patient *p* by a nurse *n* handling a mobile mediator *m*, then it has been the case that the nurse *n* had
  - identified that patient *p* using mobile mediator *m*
  - identified that device *d* using mobile mediator *m*
  - captured reading *r* from patient *p* using device *d* while still using mobile mediator *m*

In other words, correct correlation of data needs to be preserved. The EHR is only consistent when a patient's records contain reading information collected from exactly that patient. This property can be easily formulated using temporal logic.

In this case study we slightly vary the recommended task description as presented in Section 2.1 to introduce an error to help illustrate the benefit of analyzing human actions. Here the nurse executes most of the steps in the recommended task description correctly and identifies a patient and a device. But in the data collection step collects medical information from a different patient and does not notice the error. Let us now go through the steps in our framework provided in Section 3.2.1.

- Modeling human behavior:
  - The actions that can be taken by the nurse are: `identify a patient, identify a device, verify captured identities, take a reading from a device, verify the collected reading`. We add an additional initial action of associating a nurse with a mobile mediator. This action corresponds to the picking up of a mobile mediator by the nurse in the nurse's station.
  - Each nurse action is then translated into a Java module. The complete collection of Java modules is provided in Listing B.1 in the Appendix. As an example of human action, we present the code for capturing medical information of a patient using a device here:

```
public void captureDataFromDevice(MobileMediator
    mobileMediator, Patient patient, Device device){
    observedData = mobileMediator.captureDataFromDevice(
    patient, device, true);
}
```

Listing 3.1: Nurse code for capturing data from a device

- For the analysis we present a combination of nurse and mobile mediator action sequence encoded in Java in Listing A.1 in the Appendix. Lines of code corresponding to the actions performed by the nurse is presented in Listing 3.2.

```
nurse1.assignMobileMediator(mmed);
nurse1.scanPatientIDwithMobiMed(patient1, mmed);
nurse1.scanDeviceIDwithMobiMed(device, mmed);

nurse1.readScannedDeviceID(mmed);
nurse1.readScannedPatientID(mmed);
mmed.getApprovalForScannedIDs(nurse1.approveScannedIDs(mmed));
nurse1.captureDataFromDevice(mmed, patient2, device);
nurse1.readCapturedDataFromMMed(mmed);

mmed.getCapturedDataApproval(nurse1.approveDisplayedReading());
```

Listing 3.2: Nurse actions for AIDC operation

This is a scenario where there are two patients and only one device. The nurse first checks out a mobile mediator at the nurse station. Then she uses the mobile mediator to scan the RFID tag of the first patient. Then she follows the same procedure for the medical device. Then mobile mediator displays the identified entities on its screen. Then the nurse reads the displayed information and the mobile mediator collects the verification of the scanned IDs from the nurse. But as highlighted in the code, the nurse now erroneously captures reading from the *second* patient. The nurse now reads the data collected by the mobile mediator which gets displayed in the mobile mediator screen. The mobile mediator prompts the the nurse to verify scanned entities and collected data. These mobile mediator actions are indirectly prompted by preceding nurse actions and thus gets mentioned in Listing 3.2. Upon her (erroneous) approval, the data is then stored in the back-end database.

- The safety property of the AIDC scenario is:

$$\begin{aligned} \mathtt{getsStoredInEHR}(n,m,p,d,r) \quad \rightarrow \quad & \blacklozenge\mathtt{scannedPatient}(p) \wedge \blacklozenge\mathtt{scannedDevice}(d) \\ & \wedge\blacklozenge\mathtt{capturedData}(p,d,r) \end{aligned}$$

- The monitor is created for execution in JavaMOP. The Monitor code is provided in Listing C.1 in the Appendix. The monitor recognizes the identification of a patient as an event, identification of a device as an event, collection of data as an event and record insertion in a database as an event. The monitor keeps track of the patient that has been identified, the device that has been identified and the combination of device and patient that is used for medical information collection through the corresponding events. If the nurse collects erroneous information and subsequently validates the erroneous information, then the mobile mediator system will send the erroneous information to the database to be stored. The safety condition is that when a tuple of information is inserted into the EHR, only the data captured from the identified patient with the identified device is recorded in the EHR. Hence any time a record is attempted to be stored in the EHR where there is a mismatch in the patient or device involved in the identification and data collection step or the collected medical information and the information getting inserted in the EHR, erroneous tuple insertion event occurs. The monitor is asked to notify in case a violation of the safety property that no erroneous tuple insertion event ever occurs.

- When the monitor presented in Listing C.1 in the Appendix is executed along with the AIDC system model and human operator model, we find that a violation occurs when the event `ehrStoresTupleWrongPatient` occurs indicating that a tuple was attempted to be stored in the database where there is a mismatch in the identity of the patient: the patient from whom the data that was collected is not the patient who was identified.

3.4 Summary

In this chapter we have presented a methodology for monitoring a computer system being operated by a human operator for safety property violation. We have shown why a declarative specification of safety requirements is more suitable. We demonstrate that existing monitoring techniques and software can be used to model and analyze human operator influence on the safety and reliability of a system. Runtime monitoring of a system being operated by a human operator can provide insights into whether the task description of operating a system recommended to the humans is safe for the system or not. The methodology presented in this chapter can actually help evaluate safety of a system when composed with any arbitrary human task execution behavior.

Chapter 4

Modeling and Verifying Tolerable Atypical Human Behaviors

In this chapter we present the methodology to model and verify safe human operator behaviors that are claimed to be "tolerable" by the system designers. We present a technique to determine whether each and every such tolerable behavior is actually safe or not. We also present *Tutela*, our prototype tool that assists in executing the methodology. We first provide a characterization of human behavior to help reason about them.

4.1 Introduction

Irrespective of the degree of automation, computer systems need to interact with human operators. Even in highly automated safety-critical systems like aircrafts, hospitals, nuclear power reactors, ambulance dispatch systems, there are human operators guiding these systems to perform their tasks. Sometimes the humans are there to initiate and terminate the operations of these systems: the autopilot system of modern aircrafts needs to be started by the pilots to take over the flying operation of the aircrafts. Sometimes human operators are there to intervene in case of an emergency: in case an aircraft faces inclement weather, only the human pilots are allowed to intervene and perform crash-landing procedure. The responsibility of these human operators in ensuring safe operation is tremendous. At the same time it is a critical responsibility of the system designers to secure the systems against unsafe erroneous behaviors of the human operators.

Human operators are usually given a recommended task specification in order to operate a computer system. However, human operators may easily deviate from the prescribed set of actions. Their sense of judgment, susceptibility to environmental elements, unpredictable decision making patterns can introduce errors in the process of executing the recommended task sequences. Let us consider the AIDC scenario again. The goal of this scenario is to electronically identify, collect and store medical information about patients. The process is safe when every record stored in the medical database has correct association between data and patient. In order to achieve this goal the nurse needs to ensure that she did not identify one patient and take reading from another one. She should not allow incorrectly associated data and patient information to be stored in the database. The reason for the nurse being the agent responsible for correct association of data and patient is that in this scenario only the nurse is physically facing the patient and knows exactly which patient got a device attached to. A mistake may occur purely unintentionally in this scenario but result in consequences which can even be fatal. If the nurse follows the recommended task specification provided in Section 2.1, then such errors do not get introduced into the EHR. However, "to err is human". Let us consider a situation build-up here:

- Nurse N walks into a room with two patients `Bob` and `Tom` carrying a single blood pressure measuring device `BPM` that she intends to use upon both patients.
- Nurse N first identifies patient `Bob`.
- Nurse N then identifies device `BPM`
- Nurse N is suddenly summoned to the nursing station to receive a phone call.
- Nurse N leaves the room postponing the data capture process.
- Nurse N returns to the room and resumes data capture process.
- Nurse N now mistakenly uses device `BPM` to capture reading r from patient `Tom`.
- Nurse N keeps on being forgetful and approves the reading captured from patient **Tom** using device `BPM`.
- The data tuple that gets stored in the EHR is (N, `Bob`, `BPM`, r) where there is no correlation between `Bob` and r. At best the tuple could have been (N, `Tom`, `BPM`, r).

This erroneous task execution leads to an erroneous tuple getting inserted into the EHR and thereby violating the safety of the AIDC system.

Throughout a data collection sequence in the AIDC scenario, it must be the case that the nurse always correctly asserts that the patient from whom a data was collected is *the* patient whom she had identified. Intuitively: $\forall$ `patients` p_1 and p_2. `tookReadingFrom(` p_2`)` $\wedge$ `usingDevice(d)` $\rightarrow$ $\neg$identifiedPreviously(p_1).

However, one can notice that the computer system cannot provide a guarantee like this. This is a constraint that has to be guaranteed by the operator of the system: the nurse. It is the duty of the nurse to ensure that she took data from the patient she identified. No matter how many checks are included at every step, if a nurse negligently asserts that an incorrect combination of data is correct, the computer system has to accept that. This unfolding of a series of events leading to an erroneous tuple getting stored in the EHR again signifies the fact that to ensure safe operation of a system, even trusted users need to provide some "reasonable behavior" guarantees. Unless the nurse guarantees that she will always ensure that she will use the device to capture data from only the intended patient, with the current system design data integrity cannot be ensured.

Human operators are so unique in their sense of judgment, decision making capability, unpredictability that considering them to be a mere part of the environment can lead to severe consequences. Also, assessing requirement conformation of a system while getting guided by the recommended human task execution behavior is also not sufficient. A computer system needs to be verified against all possible aberrant human operator behavior. However, analyzing a system against all possible erroneous human behavior may not be feasible in any scenario. The questions that we answer in this chapter are: Can we identify the guarantee of "reasonable" behavior that human operators can provide to the computer system? How do we specify such a guarantee? What human behaviors conform to this guarantee? How do we determine whether the computer system is actually resilient against all human behaviors conforming to the "reasonableness" guarantee?

Ideally, a computer system should be resilient against all possible human judgment errors. In reality, protecting against all possible human action variations may become tedious and not cost-effective. However, there are always some important guarantees of behavior that the system expects from the human operators. If the human operators deviate from these behaviors, then the system may fail to remain safe and dependable. Hence there is a need to identify and precisely formulate the expected "reasonable" human operator behaviors. In this chapter we present a methodology for modeling and analyzing both typical and atypical human task execution behaviors.

4.1.1 Organization

This chapter is organized as follows: in Section 4.2, we present a characterization of human behavior to assist in reasoning about them. Then in Section 4.3, we present work on runtime monitoring in the presence of behavior guarantees of different operators. We present another example scenario in Section 4.4 and the human behavior characterization in that scenario. In Section 4.5 we present a framework for specifying and analyzing human task descriptions. In Section 4.6 we present our methodology for realizing the framework. We present a prototype implementing the aforementioned framework in Section 4.8.

4.2 Human Behavior Categorization

We now present a categorization of the set of all possible human behaviors using Venn diagrams in Figure 4.1. This categorization will help us clarify the notion of reasonable human operator behavior. This categorization is defined with respect to the behaviors of a complimentary set of players, in this case the computer system, the operating platform etc. For any such analysis, we need a domain-dependent concept for progress and loss. With a concept of progress and loss in place, the different categories of human behaviors are as follows:

Safe: The *Safe* behaviors are those in which the actions of the operator never lead to any *loss*.

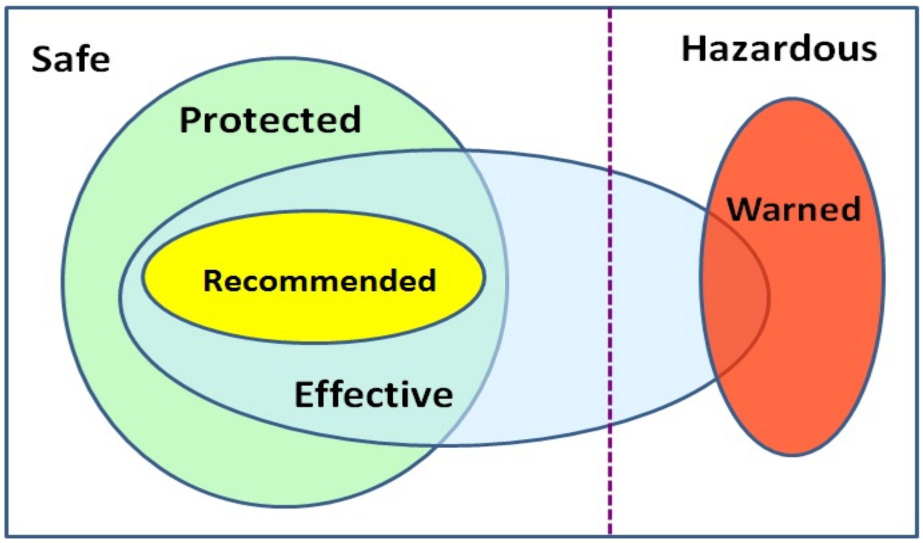


Figure 4.1: Protection Envelope

Hazardous: Each behavior that is not safe are *hazardous*.

Effective *Effective* behaviors are ones in which some progress is made.

Recommended Among the effective behaviors are some desired behaviors which we refer to as *recommended* behaviors in which the operator exactly follows the steps in his task description. There may be ways to make progress that are not recommended, perhaps because the recommended procedures are just meant to describe one of many ways to get the job done or because other ways of doing the job may be hazardous.

Warned The *warned* behaviors, often specified in the warnings section of a user manual, are the recognized set of hazardous behaviors.

Protected The *protected* behaviors are the prime focus of this dissertation. The behaviors in which the operator may vary from recommended or effective behaviors without causing any hazardous consequence are called protected behaviors. These behaviors do not have hazardous consequences because the system developers have safeguarded the system against these behaviors. The collection of all protected behaviors is referred to as the "protection envelope". They are the set of safe behaviors that have been identified to be safe by the system developers. If the humans guarantee to never behave in an unprotected manner then the system can guarantee that it will always remain safe. Thus the protection envelope is the guarantee of good behavior provided by the humans that can be relied upon by the system to ensure conformance to a requirement.

We observe that the effective behaviors may or may not be safe. While the recommended behaviors are both effective and safe, the protected behaviors are guaranteed to be safe only, they may not be effective in some situations. The "protection envelope" enforces less stringent guidelines for the human operators at the cost of a reduction in its characteristics. The protection envelope is provided by an engineered set of properties of the system that form a specified subset of safe behaviors of the human operators.

In any scenario, the designers would like to identify all possible hazardous human behaviors thereby extending the warned behaviors to become the overall hazardous behavior set. On the other hand, designers of a system would also aim at increasing the robustness of a system against atypical human behavior by enlarging the protection envelope. An idealized situation would be where the protection envelope represents all possible safe behaviors. The notion of a protection envelope can be generalized to be extended to each and every player in a scenario. Each player has to meet

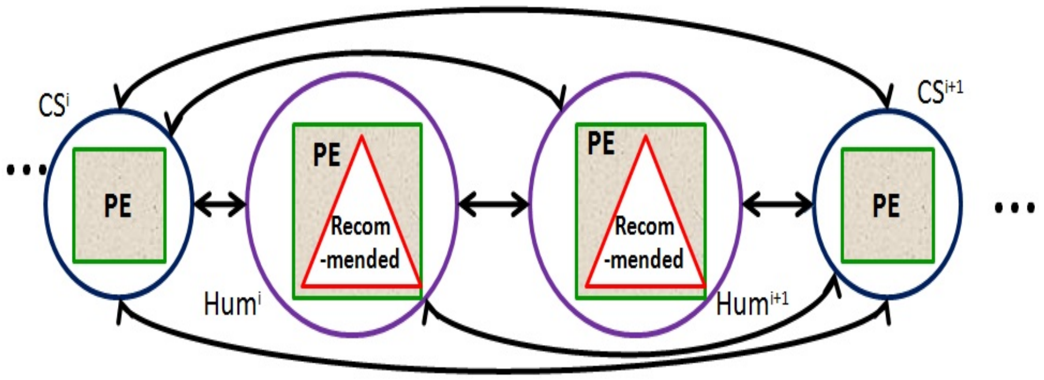


Figure 4.2: Computer System along with human operators

some requirements on their behaviors: the human operator should behave within the restrictions expected by the system developer (protection envelope); the computer system should execute to meet some requirements set by the system designer(protection envelope); the human operator may use the system along some specific step-by-step instructions (recommended task specification); the computer system will execute according to some generic specification of the system (recommended behavior).

4.3 Monitoring Safety and Protection Envelope

Every computer system component and human operators in a scenario needs to provide protection envelope guarantees to each other. Each entity may rely on the guarantees provided by the other components to ensure safe operation of the overall system. With the concept of protection envelope in place we ask the question: If a computer system being guided by multiple human operators violates its safety property, can we determine which entity was responsible for the violation? Was it a system component failure? Was it a buggy piece of code? Or was it one of the human operators operating the system erroneously? We found a suitable answer in runtime monitoring.

4.3.1 Safety Violation Blame Apportion

The framework presented in Section 3.2.1 will work for any property specified using temporal logic and a specific behavior exhibited by a human operator. The task execution behavior can be the recommended task specification for a human operator. Then if this recommended task execution behavior raises no violation alarm while being monitored, then we can state that the recommended task specification is indeed a safe one. This methodology will work for any human task execution behavior. If we perform runtime monitoring of a system combined with an arbitrary human task execution behavior and a safety property violation occurs, then we need to determine whether the violation was caused by a component of the computer system or whether it was caused by an action of the human operator. Even when a violation occurs because a system component was attempting to perform an unsafe action, we would like to know whether there was a human operator action performed earlier that led the system component to take an unsafe action.

To solve the issue raised above, we fall back to the notion of protection envelope as presented in Section 4.2. Each and every computer system component and each and every human operator needs to provide guarantees to each other. The *protection envelope guarantee*s provided by an entity is *relied* upon by the rest of the entities to ensure a requirement is satisfied. These protection envelope guarantees can be used to determine the root cause of a violation. The idea is as follows:

- Obtain protection envelopes for each system component and each human operator.
- Monitor the execution of a system with a specific human behavior using the framework presented in Section 3.2.1.
- In case of a violation of the requirement occurring during the execution of the system
  - Monitor the combination of the system and the human operators but this time monitor once for protection envelope violation of each entity.

Monitoring for protection envelope violation will hopefully provide us with the set of entities whose protection envelope guarantee was violated. An understanding of the entities who violated their expected behavior guarantees will allow us to asses them for their role in the violation. The events that triggered the monitor to report failure of a protection envelope property will hopefully provide ideas to the system designers as to why the protection envelope was violated.

4.3.2 Association of Blame in the AIDC Scenario

Although the earlier monitoring experiment determines that an attempt was made to store an incorrectly associated data tuple in the EHR, it may have happened due to some coding error in the AIDC system software or it may have been because the nurse had actually collected the reading from a patient whom she had not intended to handle. To determine what was the cause of the violation we perform monitoring again with the same human operator behavior, but first with the protection envelope of the Mobile Mediator and then the protection envelope of the nurse. The protection envelope of the mobile mediator simply states that it will take the steps expected from it: after the nurse presses the scanID button it will capture the RFID tag value of the intended entity, if the nurse validates a collected data, then it will send the data to the EHR to be stored etc. The monitor does not report any violation of the mobile mediator's protection envelope. On the other hand, the protection envelope of the nurse is the guarantee that she needs to provide to ensure correct information association in the EHR. The nurse needs to guarantee that if she verifies a collected reading from a patient p using a medical device d, then she had actually identified that particular patient p in a previous step. This protection envelope is violated by a nurse who identifies one patient and captures reading from another patient and also informs the mobile mediator that the (incorrectly collected) reading is valid for insertion into the EHR. The nurse was required to use the device on the patient she had identified. Any deviation from it violates the protection guarantee. And that is the case in the nurse behavior presented in Listing A.1. The nurse identifies the first patient but captures reading from the second patient. The monitor determines accurately that there is a violation of the protection envelope of the nurse. Thus we know that the fault lies with the human operator. However, we observe that the model of the scenario in this case allows the monitor to detect the occurrence of erroneous data collection by the nurse. In real life, the computer system can only detect this if the RFID tag of the patient is scanned again before capturing reading from a device.

4.3.3 Issues with this Approach

Runtime monitoring allows detecting run-time violation of safety and protection envelopes of the entities. Performing off-line test-runs against human behaviors will help assess the strengths and vulnerabilities of the actual system. However, the problem with this approach was that runtime monitoring or runtime analysis in general is concerned with one particular execution of the system. In our case it also means monitoring against one particular human operator behavior. In the framework provided just above, a combination of erroneous behavior exhibited by human operators can be monitored to determine which one of the operators were responsible for the violation. An issue with this approach is that of test coverage. Given a recommended task specification, we can determine whether it is (i) safe by monitoring against the safety property, (ii) protected by monitoring against the corresponding protection envelope guarantees. But how do we assert that all human operator behaviors conforming to the protection envelope guarantee are safe? How do we obtain each of these protected behaviors in the first place to analyze them via monitoring? We now present a framework of formal methodologies that will assist us to model and analyze all possible protected behaviors.

4.4 Example Scenario: Self-scan Checkout System

Before presenting our methodology we present another example scenario that we will use throughout the chapter.

Self-scan Checkout System

Everyday customers walk up to self-scan checkout systems at supermarkets to buy groceries. These self-scan checkout systems are equipped with interactive touch-screen displays, bar code scanners, bagging areas equipped with weighing scales and payment accepting devices. These equipments are controlled by a computerized system in the back end. A customer begins by pressing the word "Start". Then the system officially allows the customer to start scanning the items. After scanning, each item is placed in a bag in the bagging area where the weight of the item is matched with the expected weight. Any significant weight mismatch results in an alarm. We will ignore grocery

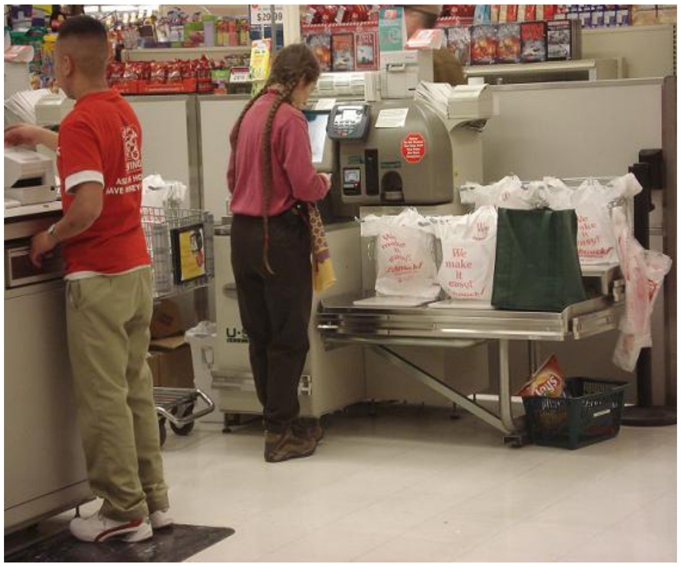


Figure 4.3: Self-scan Checkout System

items that do not have any bar codes on them. After all items have been scanned, the customer then presses the payment button, pays for the items and subsequently leaves the store with the items just purchased. Just as much as the customers can successfully follow these steps to buy items from the store, they can (un)willingly create system loss for the store owners. What if a customer scans one item but places another more expensive item of the same weight in the bag? What if the customer forgets to scan an item in the cart altogether? Having a formal, characterization of problematic and non-problematic erroneous customer behaviors will help understand, analyze and possibly improve the self-scan checkout system.

4.4.1 Behavior Categorization of Customer in Self-scan Checkout Scenario

Let us first define the notions of loss, effectiveness and safety in the context of the self-scan checkout system as presented in Section 4.4. In this context (from the store owner's point of view) a customer being able to take an item out of the store without paying for it yields loss for the store. Being able to perform any one of the actions of pressing the start button, scanning an item, putting an item in a bag or paying for an item indicates progress for the customer. The recommended human task specification is given in Figure 4.4. Let us now characterize the rest of the customer behaviors:

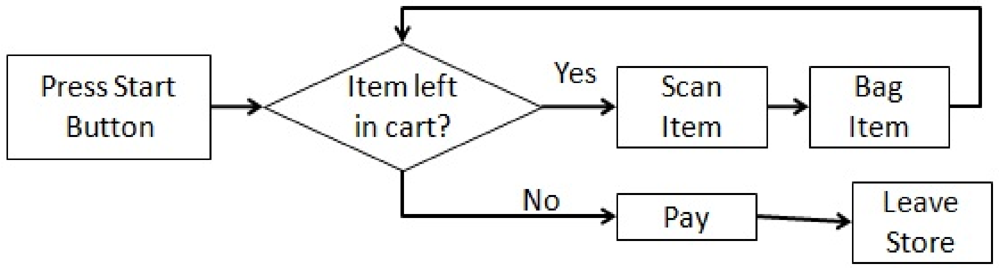


Figure 4.4: Recommended customer task specification for using self-scan checkout system

Safe: Any customer behavior that does not lead to loss for the store is a safe behavior. For example, in the very beginning, before pressing the start button a customer can attempt to bag an item. The self-scan checkout system is capable of detecting this and can generate an alarm. Hence this is a safe but not recommended behavior.

Hazardous: The hazardous behaviors are where a customer leaves the store with items that have not been purchased. If the customer leaves the store without paying for some items because they were never scanned, then the self-scan system cannot detect this. This can result in loss on part of the store and can only be avoided by the customer operating in the protected manner.

Effective: A behavior is effective if a customer is able to perform any of the actions of pressing start, scanning an item, leaving the store with items in the cart. Some effective behaviors may be hazardous: leaving the store with items that have not been paid for. Some effective behaviors are safe but not recommended. Like putting an item in the bag before scanning it.

Protected behavior: The protected behaviors are where the customers do not leave the store with items that have not been at least scanned. Once an item gets scanned the self-scan checkout system can detect whether it was subsequently bagged and paid for. For example the customer does not scan one item and put another more expensive item of equal weight in the bagging area without scanning the second item at all.

The protection envelope is the collection of “reasonable” human operator behaviors. The protection envelope provides a weakened set of restrictions on the human operators. If the human operators guarantee to at least conform to the protection envelope then the system can be demonstrated to be safe. We now present the framework for determining whether each and every protected behavior is actually safe for a system.

4.5 The Framework

We present a methodology to specify, model and analyze the behavior guarantees and requirements of computer systems and their environment components. The framework is comprised of four major steps:

1. Identify major system components and human operators and get knowledge about these elements including their observable actions from the system designers.
2. Model the behavior expected from the operators, and the properties the system should maintain.
3. Verify whether the system can maintain the properties for all reasonable operator behavior i.e. any human operator behavior that conforms to their expectations.
4. Study the robustness of the system either by allowing the human operators to deviate from expected behavior or changing their expectations.

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

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

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

4.5.1 Self-scan Checkout System Scenario Revisited

Let us now fit the self-scan checkout system described in Section 4.4 in our framework.

- Entity identification: The prominent agents are the customer and the self-scan checkout system. The observable actions of the customer are: scan an item, bag an item, pay for items etc. The behavior of the self-scan checkout system consists of actions like allow scanning an item, allow bagging an item etc. In order to ensure smooth execution of the system, the self-scan checkout system needs to store some additional information like the bar code of the items getting scanned, actual weight of items, expected weight of items, prices of items etc.
- Guarantee identification and modeling: The expected behavior (protection envelope) for the customer includes: the customer will not take out of the store items that have not been scanned. The expected behavior of the self-scan checkout system is that it will allow the customer to press the start button in the beginning; allow scanning of an item, bagging of an item; accept payment in the end etc. These properties will be modeled using temporal logic. The combined system of the self checkout system and the customer will be modeled using Concurrent Game Structures (Section 4.6).

We need to determine whether the self-scan checkout system will be able to maintain safety when facing any protected human behavior maintaining human guarantees. We discuss the third and fourth step in Section 4.6.

4.6 Methodology

The process algebraic modeling of human operators and system components as presented in Chapter 2 had both its advantages and disadvantages. CSP is capable of capturing the interactions and the communications of different agents operating concurrently. However, the procedural nature of the formalism made it difficult to succinctly express any requirement properties. The semantics of trace refinement was also not convenient. Let us take a brief recourse to the AIDC scenario to illustrate the awkwardness of specifying reasonable behavior guarantee using CSP. The nurse needs to guarantee that she will only collect data from the patient she had identified. However, there can be many different nurse action sequences conforming to this guarantee. A simple example is when a nurse is unsure that she had identified the tag of a patient correctly and scans the same patient's RFID tag again. The CSP process representing the protection envelope of the nurse will have to include all such safe behaviors. A possible such protection envelope can be:

$$\begin{aligned}
&\mathsf{NursePE}(n,m) = \mathsf{HCI}^{n}_{m}!\mathsf{GetID} \rightarrow \\
&\quad((\sqcap_{p}\mathsf{ScanPat}(n,m,p)) \sqcap (\sqcap_{d}\mathsf{ScanDev}(n,m,d))) \\
&\mathsf{ScanPat}(n,m,p) = \mathsf{HCI}^{n}_{m}!(\mathsf{RFIDChan}^{n,m}_{p}) \rightarrow \\
&\quad(\mathsf{ScanPat}(n,m,p) \sqcap (\sqcap_{d}(\mathsf{HCI}^{n}_{m}!(\mathsf{RFIDChan}^{n,m}_{d}) \rightarrow \\
&\quad\mathsf{SeeID}(n,m,p,d)))) \\
&\mathsf{ScanDev}(n,m,d) = \mathsf{HCI}^{n}_{m}!(\mathsf{RFIDChan}^{n,m}_{d}) \rightarrow \\
&\quad(\mathsf{ScanDev}(n,m,d) \sqcap (\sqcap_{p}(\mathsf{HCI}^{n}_{m}!(\mathsf{RFIDChan}^{n,m}_{p}) \rightarrow \\
&\quad\mathsf{SeeID}(n,m,p,d))))
\end{aligned}$$

$$\begin{aligned}
&\mathsf{SeeID}(n,m,p,d) = \mathsf{HCI}^{n}_{m}?x \rightarrow \\
&\quad \mathsf{if}\, x = (\mathsf{Name}(p), \mathsf{Name}(d)) \\
&\quad \mathsf{then}\,((\mathsf{HCI}^{n}_{m}!\mathsf{Yes} \rightarrow \mathsf{TakeCkReading}(n,m,p,d)) \\
&\quad \mathsf{else}\ \ \mathsf{if}\ \ x = \mathsf{Error}\ \ \mathsf{then}\,(\mathsf{HCI}^{n}_{m}!\mathsf{OK} \rightarrow \mathsf{Skip}) \\
&\quad \mathsf{else}\,(\mathsf{HCI}^{n}_{m}!\mathsf{No} \rightarrow \mathsf{Skip}))
\end{aligned}$$

In the protection envelope for the nurse, multiple scans of the identity tag of a patient or a device has the same effect as scanning it once. After a nurse has scanned a patient, a nondeterministic internal choice is made as to whether the patient will be scanned again or whether the device will be scanned as the next action. Another possible situation arises when the nurse scans a medical device's tag first and then scans a patient's tag instead of scanning the tag of the patient first and then the device's tag. This CSP process representing the protection envelope is not intuitive to create from the intuitive understanding of a protection envelope property. Let us consider the human behavior classification provided in Section 4.2. The recommended behaviors are best described in a procedural manner. However, the safe, effective, protected and hazardous behaviors are better modeled declaratively. Thus we need a formalism that provides a combination of both declarative and procedural specification of human behaviors.

We need to be able to express the human operator reasonable behavior guarantee and then assert that the combined model of reasonable human operators and computer systems satisfies a safety property. Thus we essentially want to perform model checking. Model checking is the most well established approach to software verification. Given a system model M and a property φ, the model checking question $M \models \varphi$ answers the question of whether the model satisfies the property φ. Model checking is an automata based approach. We present the formalism chosen by us below:

4.6.1 Framework in Details

Let us study the framework presented in Section 4.5. The first step suggests that the formal methodology should be able to capture different identities and their action vocabularies. In the second step, we see that the formal methodology needs to be capable of expressing the expected guarantees of behavior for each entity identified in the first step. The third step poses these problems: how do we verify that the system can maintain its guarantees in the face of all possible reasonable behavior from its human operators? What is the model for the humans and the system? How do we model and analyze all possible human behaviors? The fourth step requires the methodology chosen in the third step to be robust enough so that we can easily modify the definition of "reasonableness" or the protection envelope property and perform the same analysis without changing anything else. In summary we need a formalism that will (i) allow distinguishing among different entities (first step of framework), (ii) allow specifying the vocabulary of those entities (first step of framework), (iii) allow specifying properties on top of the formalism for each entity (first step of framework), (iv) allow modeling entity behavior (third step of framework), (v) allow modeling the interactions among the entities (third step of framework) (vi) allow reasoning about all possible reasonable behavior (third step of framework), (vii) allow efficient and simple way of re-performing the entire analysis with variations of the protection envelope. A formalism that meets these criterion is Concurrent Game Structures devised by Alur *et al* in [9]. CGS provides a way of building an automaton of human operators and computer systems where at each state, individual action choices of the entities also provide a way of specifying sub-automaton for each entity.

4.6.2 Concurrent Game Structures

A CGS is an 8-tuple, $\langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ where the components are as follows:

- P is a finite set of players.
- Q is a finite set of states and $q_0 \in Q$ is the initial state.
- We have abstract actions Σ instead of the numbers that are used as place holders for actions in [9]. Σ is a finite nonempty set of actions for the players where the silent action $\tau \in \Sigma$.
- Π is a finite set of propositional atoms that help identify the states.
- $\pi : Q \to 2^{\Pi}$ is a function that gives the set of propositional atoms that hold of a given state.

- The function $e : \prod_{p \in P} \prod_{q \in Q} 2^{\Sigma}$ gives which actions are enabled for each player in each state. We denote the action choices available to a player p in a state q by $e_p(q)$. We enforce that the silent action τ is enabled for each player in each state: $\forall q \in Q. \forall p \in P. \tau \in e_p(q)$. We will use Σ_p to denote the vocabulary of player p where $\Sigma_p = \bigcup_{q \in Q} e_p(q)$. We will denote the action choice for a player p in an action choice vector $\bar{v}$ as $\bar{v}|_p$.
- The function δ is an element of the dependent product $\prod_{q \in Q} . (\prod_{p \in P} e_p(q)) \rightarrow Q$ that defines state transitions. That is, given a state q, and a vector of actions $\bar{v} \in \Sigma^P$ where $\bar{v}|_p \in e_p(q)$ for each p, the value $\delta(q, \bar{v})$ is the next state q'. We require that $\forall q \in Q. \delta(q, \tau \times \ldots \times \tau) = q$. The reason behind enabling the τ action for each player in each state and $\bar{\tau}$ self-loops in each state is based on the perception that human operators cannot be assumed to always be synchronized with the computer systems. They may idle away for some time before performing an action.

Given a finite or infinite sequence of states λ, we will denote the i-th state in the sequence as λ_i. We will use $\lambda_{i_1}^{i_2}$ to denote the sequence of states starting from position i_1 to position i_2. A *q-computation* r of a CGS C is an infinite sequence of states where $r_0 = q$ and for each $i \geq 0$, there is a vector $a_i \in \prod_{p \in P} e_p(r_i)$ of actions such that $r_{i+1} = \delta(r_i, a_i)$.

CGS vs CSP

CSP has many advantages for modeling human behaviors. The usual semantics of CSP is in terms of process refinement. Although there has been efforts for checking traces of CSP process against temporal properties, existing CSP tools provide process refinement support. This necessitates procedural modeling of system requirements like safety properties. The procedural description of all behaviors conforming to safety requirement can be a cumbersome process. Another aspect that was related was that CSP does not have specific modularity in the sense that we cannot associate player identity with player actions like in a CGS. CSP also does not have explicit notions of states. CSP processes provide a high level description of complex action sequences thus abstracting away underlying states the processes go through.

Many human behaviors may conform to the protection envelope property. One may not be able to enumerate the protected behaviors. Thus the methodology that we will present in this section considers restricting an unrestricted human behavior model to obtain a representative protection envelope model. Once a CSP process is defined it is not easy to automatically modify it. A property conformance test is executed on the traces of execution of a CSP process. In the absence of states, it is not easy to correlate the execution traces with codes in the CSP process to modify them.

On the other hand, CGS is a collection of player modules spread over states. The action choices for the players in each states allows a procedural encoding of a behavior. Each player can make an *internal* decision about which action to choose from the actions enabled for him in each state. The action choices made by the other players are decided upon *externally* of this player. And the temporal properties that can be imposed on the players in a CGS provides a declarative technique of modeling system requirements. One can modify a CGS by modifying the components of each state. And since CGS transition system can be thought of as the transition system of automata, we can use existing model checking tools too. Thus we find that CSP is a high level and highly suitable formalism for modeling and analyzing human behavior. But for protected behavior analysis, the convenience of manipulating low level granular player modules in CGS, renders the CGS to be a more suitable formalism. Let us now fit CGS in our framework:

4.6.3 Identification and Information Collection Step

CGS will allow us to easily model all the identifiable entities, such as the computer system and the human operators as players operating concurrently. The actions available to a player across the states will capture their vocabulary. Referring to the self-scan checkout system scenario, we can have Self-scan system and Customer as the players. Since we allow abstract actions each action of the self-scan system and the customer can be translated into abstract actions. For example the actions of the customer like scanning an item, bagging an item, paying for items etc can cause the following actions to be in the vocabulary of the customer: `pressStart, scanItem, bagItem, payforItems`. Similarly, the vocabulary relevant to the self-scan system may contain: `allowPressStart, allowScan, allowBag, allowPayment` etc.

4.6.4 Protection Envelope and Safety Property Specification Step

The protection envelope is a collection of protected or reasonable behaviors. It is much easier for the system designer to provide the protection envelope using a property describing the protection criteria. Every behavior conforming to the criteria will be a protected behavior. Now we show what logic the protection envelope property can be expressed in.

The protection envelope properties are, in fact, a form of safety property. The protection envelope property for each human operator needs to state that in every state along a protected action sequence, the human operator will guarantee nonviolation of a criterion. The criterion my very well be of the form that the human operator will not perform certain task until some other enabling task has been performed. So protection envelope properties are temporal in nature, and can be expressed using Linear Temporal Logic [93]. In fact protection envelope properties are a form of safety properties [108]. Safety properties state that something bad does not happen and if there exists a counter-example to a safety property it must be a finite one. In this dissertation we will consider special form of safety properties called "invariants" [108] which are of the form $\Box\varphi$, where $\varphi = p$ (atomic proposition) $\mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \, \mathcal{U} \, \varphi_2 \mid \Diamond\varphi \mid \Box\varphi$. Invariants need to hold in the initial state and along each trace emanating from the initial trace. The protection envelope properties will also be of the form $\Box\varphi$. The protection envelope for the customer checking out items in a supermarket using LTL:

$$\Box((\neg\texttt{baggedItem}(i)\,\mathcal{U}\,\texttt{scannedItem}(i)) \wedge (\neg\texttt{itemOutofStore}(i)\,\mathcal{U}\,\texttt{itemPaidFor}(i))))$$

Here baggedItem(i), scannedItem(i), itemOutofStore(i) and itemPaidFor(i) are atomic propositions.

4.6.5 Assessing Robustness of the Computer System with Respect to the Protected Behavior

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

Human Behavior Model

The e function provides the enabled action choices for each player in each state in a CGS. Here we view a human behavior as the set of actions that the human may perform in a state. Hence, we observe that the human operator behaviors can be viewed as refinements of the e function. The scenario in Figure 4.1 is completed when the human operator is placed in conjunction with the computer system. The combined interaction between each human behavior and the computer system can be modeled using CGSs, where the behavior of the human operator will dictate the set of possible actions for the operator in each state: the e function. Now that we have a method of modeling human behavior, we need to determine how best we can analyze all possible protected human behaviors.

Maximal Model

We first need to be able to formulate all possible protected behaviors. Then we need to assess their safety preservation capability. However, formulating and analyzing the protected behaviors one by one may become cumbersome. Given only the protection envelope property, we will have to generate each protected behavior and then analyze it. Hence we ask: can we get around the task of verifying a model for each and every possible environment and find a representative model such that satisfaction of that model in conjunction with that representative protected behavior guarantees satisfaction of that property for any behavior? Grumberg *et al* suggest searching for an ordering among environments such that given an ordering relation $\sqsubseteq$, a model M, a property ϕ, if we can find a maximal model of the environment E such that if $E \parallel M \models \phi$ then for each environment $E' \sqsubseteq E$, we have that $E' \parallel M \models \phi$ [50]. The motivation behind this approach is that now given a preordering relation, a maximal model for that relation, determining satisfaction of an environment is determined by simply determining preorder preservation. Our approach very closely follows the idea presented by Grumberg *et al.* We first define an ordering relation among operator behaviors. Then we show how we can find a maximal protected behavior such that all other protected behaviors are related to it by the ordering relation. Then we show that this maximal behavior can actually represent all protected behaviors: if the maximal behavior allows the system to operate safely then all protected behaviors will also allow the system to operate safely.

Ordering Relation

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

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

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

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

4.6.6 C_{all}

We focus on the resilience of a computer system while facing atypical human behavior. C_{all} is a CGS which contains the interactions among system components and human operators where at each state, the human operators can execute any action that is physically possible by them. In every state in a C_{all}, each physically possible human action is enabled in each state. Hence the enabled actions and the transitions must contain the responses of the computer system for each possible human action. Usually at each state of a system there is a small subset of human actions that is expected by the computer system. The rest of the human actions are either physically impossible or will lead to an error state. We will present our tool in Section 4.8 which is capable of generating the C_{all} from the small subset of interactions among computer systems and human operators. We require that the description includes the safe enabled actions and corresponding reactions for the human operators, the actions that are physically impossible in each state. Any human action that is not explicitly mentioned in the specification is assumed to lead to error. Let us consider the self-scan checkout systems again. In Figure 4.5 we present a typical model provided for the initial state. In the figure, the label of each state appears next to it. Let a_1 be a computer system action and a_2 be a human action. The pair of actions (a_1, a_2) is mentioned next to the transitions they cause. In the initial state, the customers cannot pay for an item as the self-scan system displays usually do not provide any pay button at this point. This is indicated by a transition to a *dummy* state: *DisabledAction*. This dummy state is only used to help denote the impossibility of the action, it is not part of the state space. The customer may idle away the time or they may press the start button. However, the customers may also attempt to scan an item, bag an item or leave the store without scanning and paying for any item. All of these actions will result in an erroneous situation. For the sake of simplicity we use only one error state for all types of erroneous conditions. In order to reduce the burden of providing input for the entire CGS all error causing human actions can be omitted if our tool is used. The transitions emanating from the initial state in C_{all} are presented in Figure 4.6. We combine the action vectors leading to the error state in Figure 4.6 for the sake of simplicity.

A C_{all} is usually not the maximal protected behavior we need to find. Since our ordering relation is based on player action set ordering, we will need to trim player actions in C_{all} to obtain the maximal protected behavior. Intuitively, the actions of the player whose behavior is getting analyzed should be modified. Since protection envelope properties are safety properties, we should trim those actions of the player which makes him do something *bad*. The easiest way of determining "badness" is if the action choice helps the CGS land in a state where a protection property is violated.

Definition Given a CGS C, a behavior restriction φ for a player p, we define the set of bad states $\mathcal{B}$ to be the states that are unsafe or potentially unsafe: $\mathcal{B}(C, p, \varphi) = \{q \mid \eta(C, p, q, \varphi)\}$. Here $\eta(C, p, q, \varphi) = (q \not\models_C \varphi) \vee (\exists \bar{v}.\, \bar{v}|_p = \tau \wedge \eta(C, p, \delta_C(q, \bar{v}), \varphi))$.

The bad states are those states where the specification is either immediately violated or where the operator has performed such an action that he needs to rely on the other players helping him out to

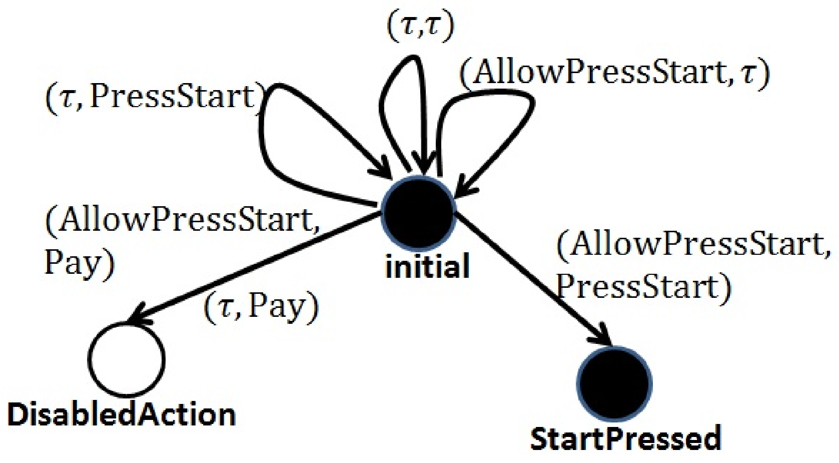
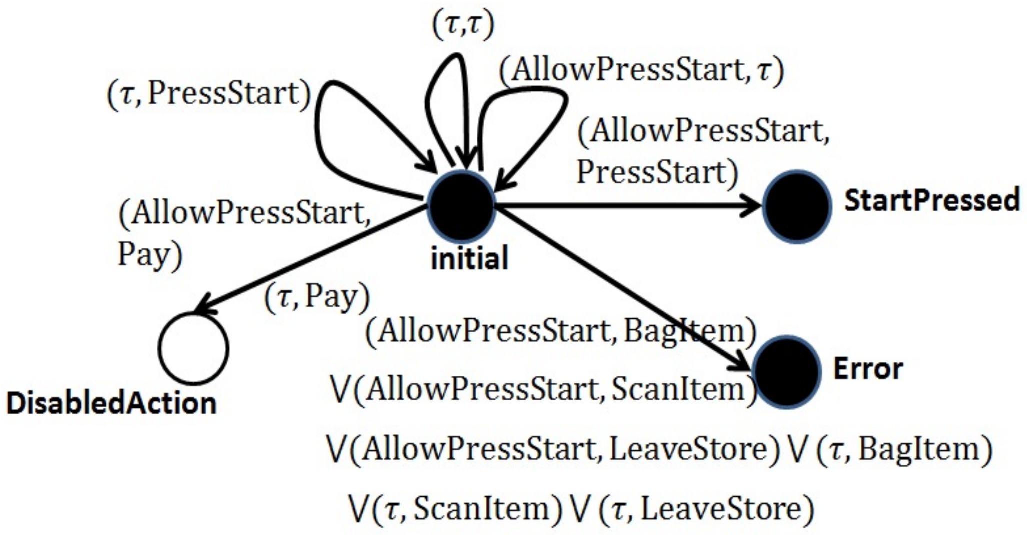


Figure 4.5: Initial state in incomplete C_{all}



(τ,τ)
(τ, PressStart)
(AllowPressStart, τ)
(AllowPressStart, PressStart)
StartPressed
(AllowPressStart, Pay)
initial
(τ, Pay)
DisabledAction
(AllowPressStart, BagItem)
∨(AllowPressStart, ScanItem)
∨(AllowPressStart, LeaveStore) ∨ (τ, BagItem)
∨(τ, ScanItem) ∨ (τ, LeaveStore)
Error

Figure 4.6: Initial state in a complete C_{all}

preserve the specification. Let us consider Figure 4.7. Here the customer has pressed the start button. Let us assume he has a non-empty cart. Then according to the recommended task specification he should scan an item from the cart. If he puts an item in the bags instead then it violates the protection property as specified in Section 4.6.4. This item has been placed in the bags before having been scanned. If the customer leaves the store with the items in his cart then items are leaving the store without being paid for. Thus both q_1 and q_3 are bad states.

4.6.7 Thinning

We will refine a given C_{all} to obtain a model representing all protected behaviors. Asarin *et al* thin actions available to a player to find a winning strategy in a game automata [13]. In a similar fashion, we "thin" the actions available to player p in an unrestricted behavior C_{all} to obtain a maximal protected behavior. The thinning process is defined below:

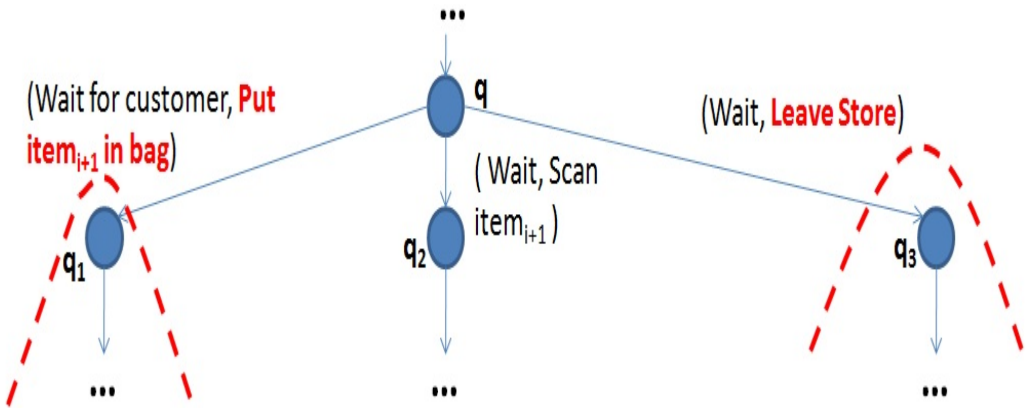

q
(Wait for customer, Put item$_{i+1}$ in bag)
(Wait, Scan item$_{i+1}$)
(Wait, Leave Store)
q$_1$
q$_2$
q$_3$
...

Figure 4.7: Thinning C_{all}

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

4.6.8 Example Scenario Revisited

Let us consider the supermarket checkout scenario. We focus on the state q after an item has been scanned in Figure 4.7. Here, the customer will have many action options in C_{all}. He can put the scanned item in a bag, scan another item, attempt to pay for the scanned item or even worse, leave the store along with the item before paying for it. The protection envelope states that the customer will not take an item out of the store before paying for it. So a protected behavior should not have the *leave_store* action enabled for the customer in state q. The mechanical infrastructure of the supermarket is incapable of preventing this action. Only the customer himself has the capability of guaranteeing that such loss for the store will not occur. Since the protection envelope of the customer is under consideration at this point and only the customer himself can guarantee safe operation, only his set of enabled actions should be reduced. This also illustrates our reasoning behind choosing CGSs over Finite State Automata. In CGSs we can very easily examine the set of actions available to players in each state and manipulate the set as necessary.

4.6.9 Thinned model is the Maximal Model

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

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

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

Lemma 4.6.1 *Let $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be a CGS. Let $p \in P$ be a player, φ be the protection envelope for p. Let Θ_C^{φ} be the CGS obtained by thinning C according to φ. Then we have* $\mathrm{Inv}(\mathcal{R}(\Theta_C^{\varphi}, q_0), \psi)$.

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

Lemma 4.6.2 *Let $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be a CGS. Let $p \in P$ be a player, φ be the protection envelope for p. Let Θ_C^{φ} be the CGS obtained by thinning C according to φ. Then each CGS C'*

such that C' is a p-subaction CGS of C and φ holds at every state reachable from q_0 in C', the portion of C' reachable from the initial state is a p-subaction CGS of Θ^{φ}_{C}. Mathematically speaking: $\forall C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle.\ \forall p \in P.\ \forall \varphi.\ \forall q \in Q.\ \exists \Theta^{\varphi}_{C} = \langle P, Q, q_0, \Sigma, \Pi, \pi, e^{\Theta}, \delta\rangle.\ \forall C'.\ (\mathcal{R}(C', q_0) \sqsubseteq_{\{p\}} C \wedge \mathrm{Inv}(\mathcal{R}(C', q_0), \varphi) \Rightarrow \mathcal{R}(C', q_0) \sqsubseteq_{\{p\}} \Theta^{\varphi}_{C})$.

Proof We will prove Lemma 4.6.2 by contradiction. Let us assume that there exists a CGS $C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e', \delta\rangle$ such that for a player $p \in P$, a property φ, $\mathcal{R}(C', q_0)$ is p-subaction CGS of C and φ is true in every state reachable from q_0 in C' but the initial state reachable portion of C' is not a p-subaction CGS of Θ^{φ}_{C}.

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

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

Theorem 4.6.3 *Let $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ be a CGS, $p \in P$ be a player, φ be the protection envelope property for p. Let Θ^{φ}_{C} be the thinned maximal protected CGS. Then each CGS C' that is a p-subaction CGS of Θ^{φ}_{C} will satisfy a safety property $\psi = \Box\psi'$ if Θ^{φ}_{C} satisfies it. Mathematically: $\forall C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta\rangle.\ (C' \sqsubseteq_{\{p\}} \Theta^{\varphi}_{C}) \rightarrow (\forall \psi = \Box\psi'.\ q_0 \models_{\Theta^{\varphi}_{C}} \psi \rightarrow q_0 \models_{C'} \psi)$.*

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

We now show how to resolve the issues raised in Section 4.5. Let us consider a human operator p whose recommended task specification has been translated into a CGS C_{rec}. Let us assume a C_{all} with initial state q_0 for the combination of the human operator and a computer system is given. Let the protection envelope property for p is `PE` and the safety property for the system is `Safety`.

- **Are protected behaviors safe?** To determine whether the computer system can operate safely while facing any protected behavior expressed by p while executing a task we need to determine whether $q_0 \models_{\Theta^{\mathtt{PE}}_{C_{\text{all}}}} \mathtt{Safety}$ where $\mathtt{Safety}$ is of the form $\Box\psi$ (ψ as defined in Subsection 4.6.4).

- **Is recommended behavior protected and safe?** Then the recommended task specification is protected and safe if we have $C_{\text{rec}} \models \mathtt{PE}$ and $C_{\text{rec}} \models \mathtt{Safety}$. However given our framework all we need to determine is whether $C_{\text{rec}} \sqsubseteq_{\{p\}} \Theta^{\mathtt{PE}}_{C_{\text{all}}}$. If C_{rec} is a p-subaction CGS of $\Theta^{\mathtt{PE}}_{C_{\text{all}}}$ then by Theorem 4.6.3 we have that $C_{\text{rec}} \models \mathtt{PE}$ as $\mathtt{PE}$ is a safety property and if the thinned maximal protected CGS $\Theta^{\mathtt{PE}}_{C_{\text{all}}}$ is safe then so is the recommended behavior CGS.

4.6.10 Experimentation with Protection Envelope

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

4.7 Find Protection Envelope?

A protection envelope for a human operator is a guarantee of "reliable" behavior. A system can rely on the protection envelope of a human operator to ensure safe operation. Given a protection envelope property and an unrestricted behavior model we can generate a CGS model corresponding to the protected human behaviors. However, what if no protection envelope has been specified? If only the safety requirement of a system is present, then we can *thin* the actions of a human operator every time the safety property is violated instead of the protection envelope of that operator. Instead of conforming to the protection envelope the human operator here needs to conform to the safety property. Any violation of the safety property is then attributed to this human operator. One can even observe that the reliable behavior guarantee of the human operator is now specified coarsely. The safety property imposes lesser restrictions on the human operator actions except for the case where the protection envelope coincides with the entire safe behavior region as depicted in Figure 4.1.

4.8 Tutela

We are building a prototype that implements the proposed framework that we call "Tutela". Its functionality in light of the framework presented in Section 4.5 is as follows:

4.8.1 Identifying and Modeling Important Players

Tutela provides a GUI to identify the major players in a scenario and build a C_{all} that models their interactions. It allows creation of a repository of the domain knowledge about the various components of a scenario. These include identifying the players of interest, their vocabulary, the guarantees expected from their behaviors, the variables they need to modify to capture the effect of their behavior on the environment. Tutela allows these collections to be dynamically modified at any stage. Tutela

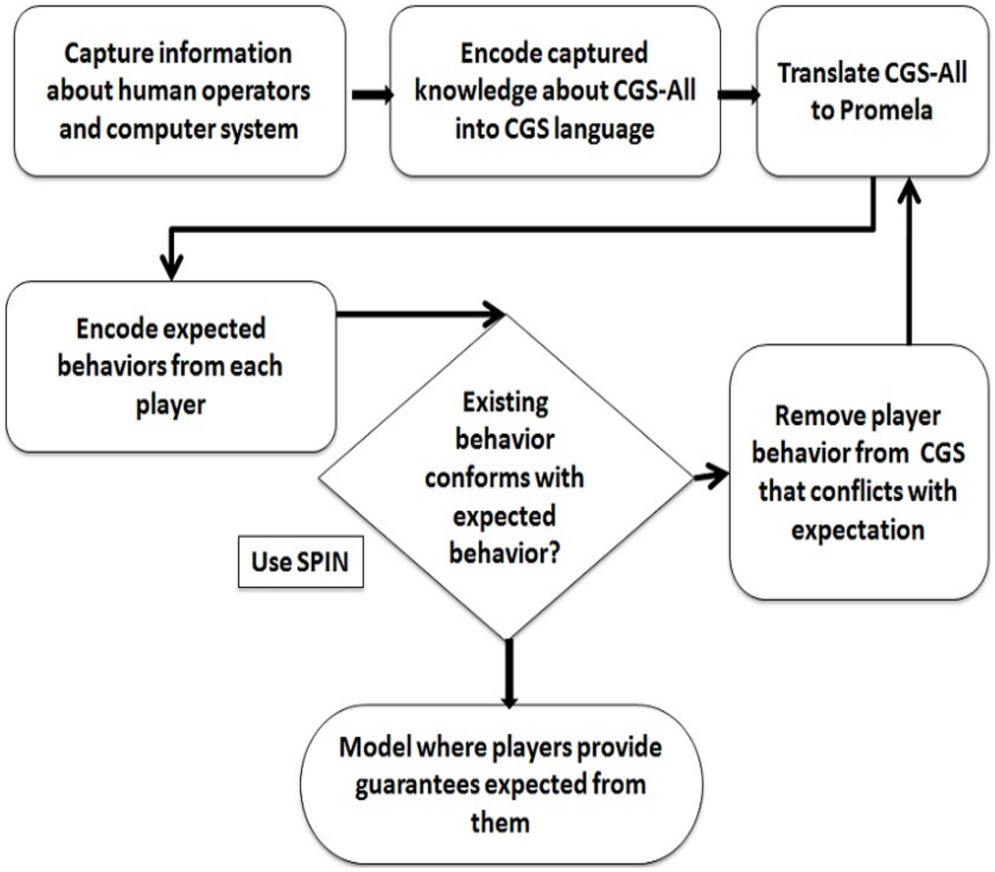

Capture information about human operators and computer system
Encode captured knowledge about CGS-All into CGS language
Translate CGS-All to Promela
Encode expected behaviors from each player
Existing behavior conforms with expected behavior?
Use SPIN
Remove player behavior from CGS that conflicts with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

Figure 4.8: Flow chart for Tutela

assists in building $C_{\text{all}} = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$. Tutela offers a GUI to allow creation and modification of Σ and Π. It also allows creation of the set of variables that are handled by players in C_{all}. The state space of C_{all} is built in an incremental fashion. Each state needs at least the propositions that will identify it and the enabled vocabulary for each player. Tutela ensures that τ is in the vocabulary of each player in each state. At each state, disallowing a human action indicates domain knowledge on the part of the designer about physical impossibility of that action. A major component of each state is the transitions emanating from a state. We allow new states to be added as needed while creating the set of outgoing transitions from a state. This allows incremental build-up of a CGS. For each state, all possible allowed action combinations need to be handled. Also the variable assignments can be specified at each state. After all required transitions for all states have been specified, the CGS is encoded into an intermediate language which can be translated into the input language for a model checker. The BNF grammar for this CGS specification syntax is:

CGS := *begincgs* cgsexpr *endcgs*

> cgsexpr := predicatesexpr plyersexpr allplayeractionexpr statelistexpr
>
> predicatexpr := *predicates* stringlistexpr
>
> playerexpr := *players* stringlistexpr
>
> allplayeractionlistexpr := *beginallplayeractionlist* playeractionlistexpr *endallplayeractionlist*
>
> playeractionlistexpr := *beginplayeractionlist* string stringnonemptylistexpr *endplayeractionlist*
>
> statelistexpr := *beginstatelist* state_expr list *endstatelist*
>
> state_expr := *beginstate* string playeractionlistexpr validpropositionexpr transitionlistexpr *endstate*
>
> validpropositionexpr := *beginvalidpropositions* propositionlistexpr *endvalidpropositions*
>
> transitionlistexpr := *begintransitions* transitionlistexpr *endtransitions*
>
> propositionlistexpr := $\perp$ | string ; propositionlistexpr
>
> transitionlistexpr := string , string , string ; transitionlistexpr
>
> stringlistexpr := $\perp$ | string ; stringlistexpr
>
> stringnonemptylistexpr := string ; stringlistexpr

Alternate Method of Creating C_{all}

The above interactive method of creating C_{all} can become cumbersome. As mentioned in Section 4.6.6, we can automatically generate C_{all} from a restricted version of C_{all}. Our tool accepts sparse C_{all}s written using the Promela language. Promela is the input language for the model checker SPIN [59]. There are some necessary information that must be present in the incomplete C_{all}: (i) the set of state propositions, (ii) vocabulary of the system and the human operator, (iii) the state space with enabled propositions, (iv) physically impossible human actions, (v) enabled system actions and human actions should be mentioned. Any system action that is not explicitly enabled will be considered to be disabled in the state. Once the enabled action sets for the human and the computer system have been established, the transition system of C_{all} is populated with all specified transitions involving them. Any transition that is not explicitly mentioned in is assumed to lead to an error state. We studied with a simplified self-checkout system scenario where the customer and the checkout system had six actions each. There were six states. Thus there were $6\times6\times6 = 216$ possible transitions in total. However, in most of the states most of the checkout system actions were physically impossible. Like in the initial state, the checkout system does not accept payments and can not process scanning of an item. Out of the 216 possible transitions 166 were disabled in this scenario. In the initial state, one customer action and four checkout system actions were physically impossible. Thus out of the 36 possible transitions, only 10 were physically possible. Out of the ten possible transitions six led to the error state. Like, attempting to bag an item before pressing the start button. Thus with sparse C_{all} specification method, one only needs to specify only the remaining four meaningful transitions from the initial state instead of all ten possible transitions. An example sparse C_{all} and the complete C_{all} is provided in the Appendix in Listing D.1 and Listing D.2 respectively. The Promela encoding of C_{all} is translated into our intermediary language for use in the next steps.

4.8.2 Modeling Protected Behavior

Currently one needs to formulate the protection envelope property to Tutela by themselves. In future we intend to augment Tutela with a tool like Propel [109] to have a guided property creation interface. Once C_{all} has been created, we can thin C_{all} with the protection envelope formula to arrive at a model that captures only the protected human behaviors and the computer system's response to them. We can view C_{all} as a finite state automaton where the state space remains the same and the transition function gets modeled by labeled transitions in the automaton. We can perform LTL model checking on this automata to check for protection envelope property satisfaction. Tutela uses the LTL model checker SPIN to help in thinning the C_{all} based on the protection envelopes of the players. C_{all} is thinned by Tutela by implementing an approximation of the definition in Section 4.6.7 by taking the following steps:

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

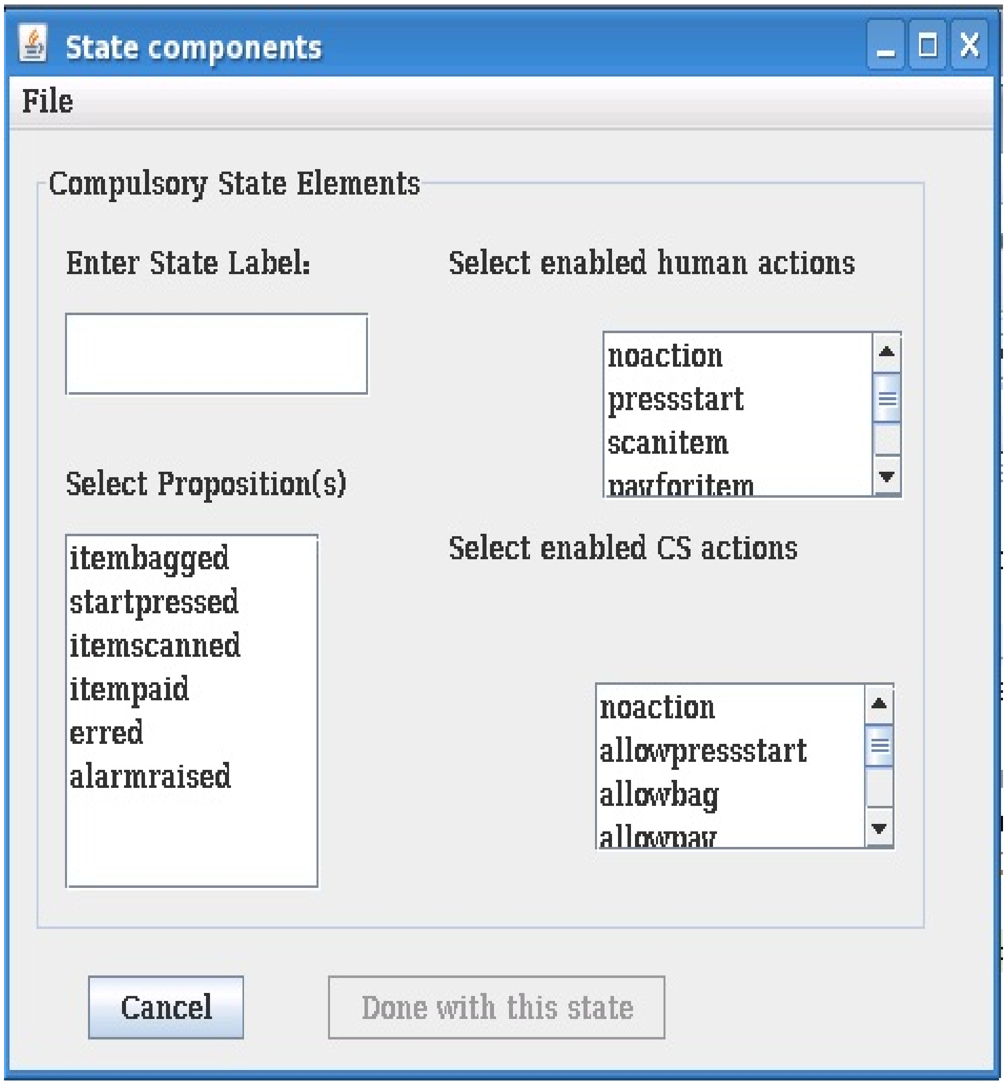


Figure 4.9: Screen shot from Tutela

- Generate scripts to execute SPIN to determine whether the CGS satisfies the protection envelope. Execute those scripts.
- Analyze the output from SPIN to find whether there is a violation of the protection envelope property. If there is a violation, automatically determine the last transition from the counter example generated by SPIN. That transition indicates the action that need to be deleted from a player's available actions. For example, in the self-checkout model we studied, the customer can put an item in the bag in the initial state before even pressing the start button as shown in Figure 4.6. The proposition indicating that the start button has been pressed is false in the error state and the proposition indicating that an item has been bagged is true in the error state. Thus the error state is a bad or unsafe state. Hence Tutela removes the `BagItem` action from the list of actions enabled for the customer in the initial state in the protection envelope CGS.
- The last two steps are repeated as many times as needed until no counter example trace starting from the initial state can be found.

4.8.3 Recommended Task Specification Model

A CGS will be used to internally model the recommended method where the recommended steps dictate the enabled human operator actions in each state. At this point we offer two methods of providing the CGS corresponding to the recommended method. The first one uses the CGS creation GUI to create the CGS corresponding to the recommended behavior, C_{rec}. The other one transforms a sequence of human actions into a CGS. This method requires that the C_{all} be already created.

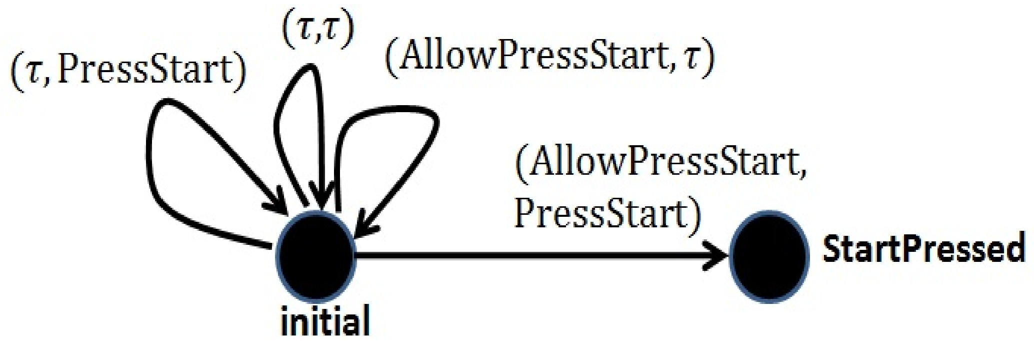


Figure 4.10: CGS for recommended way of executing task

Then each human action in the recommended task execution behavior allows Tutela to determine the enabled action function and the transitions starting from the initial state. For example, given a recommended task guidance : `PressStart, ScanItem` and the initial state of of C_{all} as presented in Figure 4.6, the the initial state of the CGS for the recommended behavior will be as presented in Figure 4.10. An example simple recommended task specification and the CGS extracted from a C_{all}(Listing D.2) using it is presented in the Appendix in Listing D.3 and Listing D.4 respectively.

Performing Our Framework Steps

After creating protected behavior CGS C_{PE}, one can verify that it satisfies the safety property via model checking. Then, given a C_{rec} and a C_{PE} Tutela can help determine whether $C_{\text{rec}} \sqsubseteq_{\mathcal{P}} C_{\text{PE}}$ to see if C_{PE} is safe too. Once C_{all} has been created, one can experiment with it by using different protection envelope properties to assess the degree of robustness of the computer system against atypical human operator behavior. In future, we intend for Tutela to (i) automatically assist in assessing the robustness of the computer system against common human errors suggested by Hollnagel [58], like repetition, out of order execution, omission; (ii) automatically further assist in building C_{all}. For example: using user specified boolean propositional restrictions to automatically generate the actions available to players in each state. Given a CGS $C = \langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ and a propositional formula φ over propositions in Π for an action a of player p, we translate it into an automaton $C' = \langle P, Q, q_0, \Sigma, \Pi, \pi, e', \delta' \rangle$ as follows: for each state $q \in Q$, if $q \models \varphi$ then $a \in e'_p(q)$ otherwise $a \notin e'_p(q)$. δ' is defined as: $\forall q \in Q.\ \forall \bar{v} \in e_{p_1}(q) \times e_{p_2}(q) \times \ldots e_{p_{|P|}}(q).\ \delta'(q, \bar{v}) = \delta(q, \bar{v})$.

4.9 Summary

In this chapter we have provided a uniform modeling approach towards human behaviors and computer systems. We have provided methodology for modeling different categories of human behaviors. The principal contribution of this chapter is a generic framework for modeling and verifying protection envelope conforming human operator behaviors.

Chapter 5

Protection Envelope Robustness Analysis

Once a protection envelope has been provided by the system designers, there arises the question of whether the protection envelope is a sufficient one. In this chapter we provide a framework for assessing the *quality* of protection envelopes.

5.1 Introduction

A protection envelope is a measure of the robustness of the automated system against human action variations. The system is robust against a human behavior if that behavior does not lead to non-safety. Ideally each and every aberrant human action variation should be a protected one. Since it is usually infeasible to identify and protect against all possible atypical human behaviors, a trade-off is needed. Usually human operators do not deviate too much from their recommended procedures. Sudden change in working environment, ambiguously designed user interface, inattentiveness, error detection failure of equipments can cause human operators to deviate from the recommended procedure and perform erroneous actions. An erroneous human action is an action that causes the system to land in a state different from the expected state. A protection envelope of a human operator is a guarantee by that human of refraining from specific erroneous behaviors. A protection envelope property may very well be satisfied by some human action sequences containing erroneous actions. These erroneous actions are protected because they do not cause disastrous effects on the system. Even so, these behaviors get classified as erroneous because they deviate from the recommended task sequence in some way. However, they are still protected because the system can safeguard itself against them. As a heuristic measure of the *quality* of the protection envelope, we take help of the existing research on characterization of erroneous human actions. A prime focus of such work is on the reasons humans perform erroneous actions. We will instead focus on the patterns of erroneous human operator behaviors not the reasons behind the erroneous actions.

5.1.1 Existing Classifications of Erroneous Human Actions

There has been numerous work on classifications of human errors. Rasmussen [96] presented three categories of human behavior: skill based, knowledge based and rule based. This classification is the basis of error categorization in many modern works. Norman categorized human errors as slips (unintentional) and mistakes (intentional) in [88]. Reason provides the “Swiss Cheese” model for errors where both the human erroneous actions and the systems which the humans operate can combine to cause more errors in [97]. Most of these research focuses on the reasons behind the erroneous actions, not on the patterns of errors. We are in search of the common variations displayed by humans executing the steps in a task. Swain *et al* proposes four human error categories: (i) errors of omission, (ii) errors of commission, (iii) sequence error (out of order execution), (iv) time error: performing actions too fast, too slow etc. in [114]. Apart from this work Hollnagel’s phenotypes of erroneous actions [58] provide us with a thorough listing of usual patterns of atypical task executions by humans. We present it in the next subsection.

5.1.2 Hollnagel’s Characterization of Erroneous Human Actions

Task analysis provides a hierarchical decomposition of a task into several subtasks. Hollnagel characterizes the erroneous actions humans may perform while executing the subtasks or actions belonging to a task. The characterization is comprised of two broad categories: *phenotypes* and *genotypes*. The genotypes are based on the reasons leading to erroneous actions and is not considered by us. The phenotypes of erroneous actions are based upon the way erroneous actions gets manifested. Then based on the level of detection he provides three classes of erroneous actions:

- Zero-order detection: The erroneous actions that can be detected by comparing with an expected action or a prescribed action sequence.
  - Zero-order erroneous action detections can be categorized based on the timing involved in their execution:
    - Premature start of an action: A subtask is started too early.
    - Delayed start of an action: A subtask is started too late.
    - Premature finishing of an action: A subtask is finished much earlier than the expected finishing time.
    - Delayed finishing of an action: A subtask is finished much later than the expected finishing time.
    - Omission of an action: A subtask is never started.
  - Sequence-based phenotypes have also been offered:
    - Jump forward: An action much later in a task sequence is performed early.
    - Jump backward: An action much earlier in a task sequence is performed later.
    - Omission: This is actually a repeat of the time-based omission phenotype presented earlier. Here an action in exactly the next step than the currently expected step is executed.

        * Repetition: An action in a step is executed a second time.
        * Intrusion: An action from another task sequence gets executed.

- First-order detection: Detecting first-order erroneous actions involves combinations of several zero-order erroneous actions and may need to consider history of zero-order erroneous actions that have been performed.
    - Spurious intrusion: In this a possibly unrelated or inappropriate action inserted in a task description without any reason.
    - Jump, skip: Several steps may be omitted or performed ahead or performed later.
    - Place loosing: A more involved permutation of the actions in a task sequence almost giving the impression of random execution of task.
    - Recovery: A forgotten action or a group of actions may get performed.
    - Side-tracking: An expected part of the task gets replaced by another part of the task. A special case is *restart*.
    - Capture: Here a part of another task sequence is inserted in place of a part of the desired task sequence. A special case of capture is *branching*.
    - Reversal: It is the permutation of two adjacent actions: constituted of a jump forward and jump backward of actions.
    - Time compression: A sub-sequence of actions gets performed in a time much less than their combined expected execution time.
- Second-order detection: Detection of second-order erroneous actions involves detection of nested or limited number of first order detections.

This deviant human task execution characterization provides us with a guideline as to how to test the robustness of a protection envelope. Every interactive system may need to be capable of remaining safe against these standard behavior variations. This idea is very similar to software testing. In this case we focus on testing the computer system software against human action variations. Studying variations of the recommended task specification according to such well established error patterns can help obtain a reasonable understanding of the strength of the protection envelope. If a protection envelope contains usual human action variations then we state that the protection envelope does a *quality* job of tolerating all standard human action variations.

5.1.3 Contributions

We provide a framework that will help analyze a protection envelope if one is present [126]. However, if no protection envelope has been formulated yet, the framework will help identify a possible protection envelope by studying the robustness of the computer system against erroneous human behaviors.

- **Protection envelope present: We can test its robustness.** The questions that we answer are: Given a protection envelope and a recommended task description, are the standard atypical executions of the recommended task contained within the protected behaviors? More specifically, is the protection envelope capable of protecting the system from typical human operator action variations as suggested by Hollnagel? If not, what are the erroneous behaviors that are not covered by the protection envelope? The system designers should at least be aware of the specific atypical behaviors that can compromise safe operation of the system.
- **Protection envelope not present: We can suggest what it might look like.** Even in the absence of a protection envelope, we should be able to assess which variations of the recommended behavior are safe. This is the second issue that we resolve. There are situations where system designers develop a system without striving to make it robust against the environment. There can be cases where determining the expectations of the system about human action variations is rendered unimportant by the extreme difficulty of developing a simple non-robust system. There are situations where precisely formulating the protection envelope is not an easy undertaking. In these types of situations the developers are tempted to design a system without putting any thorough emphasis on the environmental effects. In all these scenarios although having a protection envelope would still be desirable, it will not be available for analysis. In such situations, we still need to assess the robustness of the system against all standard human action variations. This can be achieved by falling back to the safety properties. We can assess whether standard varied action sequences conform to the safety property instead of the protection envelope. The advantage of our study in this case will be that the analysis results will help identify the safe aberrant human behaviors. We will provide a collection of human behaviors deviating from the recommended tasks specification that can be demonstrated to be safe. This collection of deviant but safe human behaviors is in fact *a* protection envelope.
- We present an extension of Tutela that can perform the robustness analysis.

5.1.4 Organization

In Section 5.2, we present the framework to assist in evaluating robustness of a protection envelope. We present extension of Tutela that helps in implementing the framework in Section 5.3. In Section 5.4 we present our findings in the study of the AIDC scenario with the extension of Tutela implementing the framework.

5.2 Framework for Analyzing Operator Action Variations

The framework we propose is comprised of four steps:

1. Construct a model for the interactions among the automated system components and the human operators.
2. Capture the recommended task specification for the human operators and safety and protection envelope definitions.
3. Introduce well established variations into the recommended task specification.
4. Perform analysis of the various mutant task specifications and provide results to the system designers about safety conformance or containment in protection envelope for each of the mutated task specifications.

We now need to resolve the following issues: How to encode the interactions between the human operator and the computer system? How to model the recommended task specification? How to introduce variations into the recommended task specification? How to analyze the different transformed tasks?

5.2.1 Human Computer Interaction Model

We need a model that can identify different entities, the actions performed by different entities in different situations and is able to capture the evolution of the system through the combined actions of the entities. The model offering all these characteristics is the *C*oncurrent *G*ame *S*tructures(CGSs) as presented in Section 4.6.2. A CGS is an 8-tuple, $\langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$. Here P is the set of players. Q is the set of states. q_0 is the initial state. Σ is the action vocabulary of the players. Π is the set of atomic propositions. π labels the states with the propositions that hold in that state. e provides the actions enabled for each player in each state and δ is the transition function.

5.2.2 Human Computer Interaction Model using CGS

In any scenario under consideration, the human operators and the computer system can constitute the set of players. The vocabulary of these players can be captured by Σ. The enabled action function e can indicate which actions are physically possible for the human operators to perform. The combined actions from the operators and the computer system can help the system transition from one state to another. Thus the transition function δ will capture the interactions among the system components and the human operators. In the AIDC scenario, the players can be: the nurse, the patients, the devices, the RFID tags of the patients and the devices, the medical mediator and the EHR. The nurse and the medical mediator can have actions like: {`PressStart`, `ScanPatID(i)`, `ScanDevID(i)`, `VerifyIDs`, `TakeReadingFromDev(p,d)`, `VerifyDisplayedReading`} and {`AllowPressStart`, `AllowScan-PatID`, `AllowScanDevID`, `DisplayScannedIDs`, `CaptureReadingFromDevice(d)`, `Display-CapturedReading(t)`,...} respectively. The collective action set of the CGS will be the union of the action sets of each player in a scenario. In the initial state, the nurse is allowed to press a start button to initiate the AIDC process. Then in the initial state, the enabled action function *e* for the nurse and the medical mediator will contain the actions `PressStart` and `AllowPressStart` respectively. Let there be a state *AIDCstarted* where a proposition *PressedStart* is true indicating that the start button has been pressed. From the initial state, if the nurse and the medical mediator performs the actions `PressStart` and `AllowPressStart` then the system can go to the *AIDCstarted* state.

5.2.3 Recommended Task Specification

Recommended procedures will be a sequence of human operator actions where the actions will be in the action set Σ of any CGS corresponding to the procedure. Consideration of recommended task descriptions with more complex control flow is a future direction. For example a recommended task description for the nurse for taking reading from a patient can be a sequence of actions: `PressStart`, `ScanPatID(i)`, `ScanDevID(j)`, `VerifyIDs`, `TakeReadingFromDev(i,j)`, `VerifyDisplayedR-eading`.

5.2.4 Protection Envelope Model

The protection envelope is essentially assumptions or restrictions placed on the human operator by the system designers. The operators cannot violate these restrictions and still be protected. The protection envelopes can be most efficiently and succinctly specified by using logical properties. Each human operator behavior satisfying the property will be a protected behavior. Since the protection envelopes will involve guarantees like "Step B is *not* taken *until* step A has been finished", the properties will need to be temporal in nature too. We will again use Linear Temporal Logic [93] formulae for this purpose. LTL formulae are usually checked against finite state automata. The transition system of a CGS is similar to the transition system of a finite state automaton with a labeled transition system. Specifying the protection envelope using LTL will allow us to verify inclusion of a behavior in the protection envelope by simply checking whether the CGS corresponding to that behavior satisfies the protection envelope property. Similar reasoning works for specifying safety properties using LTL. Also the existence of efficient LTL model checkers is an added incentive for this approach. In the AIDC scenario, the nurse is responsible for every pairing of patient ID, device ID and data that is stored in the EHR. The nurse validates the IDs scanned from the patients and the devices. A part of the protection envelope for the nurse can be:

$$\square(\mathtt{patIDOked(i)} \wedge \mathtt{devIDOked(j)} \rightarrow \mathtt{patIDScanned(i)} \wedge \mathtt{devIDScanned(j)})$$

Here `patIDOked(i)`, `devIDOked(j)`, `patIDScanned(i)` and `devIDScanned(j)` are CGS state propositions. They indicate that the nurse has just now verified *i*-th patient and *j*-th device's IDs and at some previous step *i*-th patient and *j*-th device's IDs were scanned respectively. We have a simple model where once a proposition is set to be true, it is set to be true for the subsequent states. For example after the i-th patient has been scanned, the proposition `patIDScanned(i)` holds of subsequent states where a device is identified, readings are captured. After one execution of the AIDC system finishes, the system returns to the initial state where all propositions are false. This allows to work with LTL formulae instead of PTLTL property.

A nurse while undertaking training for using the medical mediator might ask whether she can scan the RFID tag of one patient multiple times. What if she scans the RFID tag of the intended device before scanning the RFID tag of the intended patient? Will she have to restart the whole process if she makes this type of mistake? It is not readily comprehensible from the provided protection envelope whether it contains such variations. This illustrates yet another utility of our framework: a situation where the protection envelope is too convoluted to explicitly express closure under different human action variations. Our framework will be capable of providing the answers to such queries.

5.2.5 Obtaining Deviations from Recommended Procedure

The characterization provided in Section 5.1.2 is quite extensive. We consider zero-order and first-order detection in our work.

- Zero-order detection:
  - Zero-order erroneous action detection categorization based on the timing involved in their execution:
    - Premature start/end of an action and Delayed start/end of an action: We do not have a timed model and hence are incapable of directly analyzing such modifications. However we will show later how delayed task execution can be approximated due to allowing each entity to stall temporarily using the silent τ action in each state in our CGS models.
    - Omission of an action: Let a task T be linearly composed of n subtasks T_i: $T = T_0, T_1, \ldots, T_{n-1}$. Then for each subtask, we can create a new task specification by omitting it. For example: $T' = T_0, T_2, \ldots$. Then we analyze the task specification to determine protection envelope containment.
  - Sequence-based phenotypes:
    - Jump forward and backward: Let a task T be linearly composed of n subtasks T_i, $T = T_0, T_1, \ldots, T_{n-1}$. Then for each subtask T_i we insert it in front of all other subtasks to create new task specifications and analyze them. For example: for subtask T_1, we consider $T' = T_1, T_0, T_1, \ldots, T_{n-1}$, $T' = T_0, T_1, T_1, T_2, \ldots, T_{n-1}$, ..., $T' = T_0, T_1, T_2, T_1, \ldots, T_{n-1}, T_1$. Then we analyze all of these mutated task specifications for possibility, validity and safety. Note that with our method of handling

jump forward and backward, the number of subtasks in the modified task descriptions increases by one.

  - Omission: The method for creating new task specification due to omission of a task is the same as omission for the timing based phenotype.
  - Repetition: Let a task T be linearly composed of n subtasks T_i: $T = T_0, T_1, \ldots, T_{n-1}$. Then each subtask is repeated once to create new tasks $T' = T_0, T_1, \ldots, T_i, T_i, \ldots, T_{n-1}$. Then each new task specification can be analyzed to determine whether this is physically possible and safe or protected.
  - Intrusion: Given two tasks $T = T_0, T_1, \ldots, T_n$ and $T' = T'_0, T'_1, \ldots, T'_{n'}$ we obtain new tasks T'' by inserting each subtask $T'_i \in T'$ after each subtask T_j in T. For example: $T'' = T_0, T'_0, T_1, T_2, T_3, \ldots, T_n$.

- First-order detection: Detecting first-order erroneous actions involves combinations several zero-order erroneous actions and may need to consider history of zero-order erroneous actions that have been performed.

  - Spurious intrusion: Let a task T be linearly composed of n subtasks T_i, $T = T_0, T_1, \ldots, T_{n-1}$. Let a be an unrelated action not even belonging to another task description. Then we can insert this spurious action in front of each subtask to create mutated task descriptions as follows: $T = a, T_0, T_1, \ldots, T_{n-1}$, $T = T_0, a, T_1, \ldots, T_{n-1}$, ...
  - Jump, skip: Let a task T be linearly composed of n subtasks T_i, $T = T_0, T_1, \ldots, T_{n-1}$. Then meaningful omissions will omit i actions where $1 \leq i \leq n-1$. For each of these cases, we can create a new task specification like: $T' = T_0, T_3, \ldots$, $T' = T_0, T_4, \ldots$ and so on.
  - Place loosing: One possible way of testing against place loosing is considering each and every permutation of the actions in a given task sequence. Given a task containing k actions, this process will have $O(k!)$ complexity though.
  - Recovery: To some extent, recovery is covered by the pairwise permutation of actions.
  - Side-tracking: We have not considered side-tracking as of yet.
  - Capture: Given two tasks $T = T_0, T_1, \ldots, T_n$ and $T' = T'_0, T'_1, \ldots, T'_{n'}$ we obtain new tasks T'' by inserting each subsequence of task T' after each subtask T_j in T. For example: $T'' = T_0, T'_0, T'_1, T_1, T_2, T_3, \ldots, T_n$.
  - Reversal: Let a task T be linearly composed of n subtasks T_i: $T = T_0, T_1, \ldots, T_{n-1}$. Then for each subtask T_i we permute it with all other subtasks and create new task specifications. For example: for subtask T_1, we consider $T' = T_1, T_0, \ldots, T_{n-1}$, $T'' = T_0, T_2, T_1, \ldots, T_{n-1}$, ..., $T''' = T_0, T_2, T_1, \ldots, T_{n-1}, T_1$ and so on. Then we analyze all of these permuted task specifications.
  - Time compression: We do not consider this type of erroneous action at this point in time.

5.2.6 Analyzing Mutant Task Specifications

We introduce a broad class of variations into the recommended task description to constitute new task descriptions. We first need to build CGS models of human computer interactions for each new task description to be able to analyze it. Each new task corresponds to a human behavior. The behavior is exhibited by executing steps of the task while interacting with the computer system in the specific order suggested by the task specification. Using this view of human tasks we describe how we can create CGSs corresponding to the varied task descriptions.

5.2.7 Model for Recommended Procedure and its Variations

In order to facilitate the process of building a new CGS for each atypical behavior created from the recommended behavior, we first construct a CGS we refer to as the C_{all} as presented in Section 4.6.6. In all states in the C_{all}, each and every action that is physically possible for the operators to execute along with the response of the computer system (and hence all corresponding transitions) will be enabled in the e function. Referring to Figure 4.1, C_{all} contains all possible safe and hazardous human behaviors. Given a C_{all} and a task specification, we will create the CGS corresponding to the task specification by constricting the C_{all} by restricting the actions available to the human operator at each state. If the task represents a safe behavior, then this process will remove hazardous behaviors from the C_{all}. The first subtask of the task specification will be searched for in the initial state. All other human operator actions (except for τ) will get disabled in that state. From then on, the subtasks will help restrict the e function for the operators. Starting from the initial state, the transition induced by the recommended operator action will be followed to ascertain the next state for inspection. At every state reached by following the recommended tasks specification, the e function and hence the transition function δ will be restricted. If at a state in C_{all}, the operator action suggested by the task specification cannot be performed then we know that the task specification is not a feasible one.

5.3 Analyzing Aberrant Behavior CGSs

Provided with a C_{all}, a mutant task specification, and a protection envelope or safety property, we can construct the CGS corresponding to that task specification using the procedure as described in Subsection 5.2.7. We then need to analyze the CGS with a suitable model checker. We have developed an extension to a tool Tutela that assists in analyzing recommended task specification variations. The tool works as follows: it

1. takes a C_{all} and a recommended task specification as input,
2. automatically
   (a) derives new task specifications from the recommended task description,

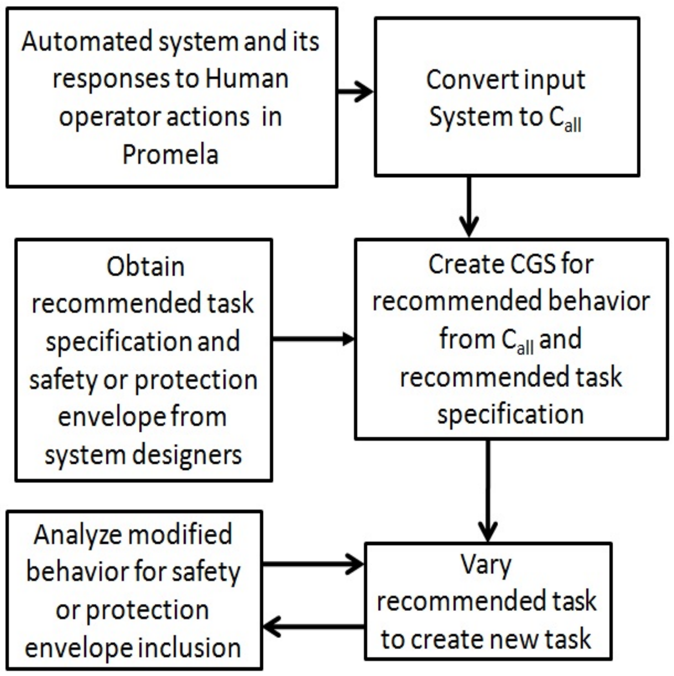

Automated system and its responses to Human operator actions in Promela
Convert input System to C_{all}
Obtain recommended task specification and safety or protection envelope from system designers
Create CGS for recommended behavior from C_{all} and recommended task specification
Analyze modified behavior for safety or protection envelope inclusion
Vary recommended task to create new task

Figure 5.1: Framework implementation in Tutela

(b) creates CGS corresponding to each new task, and

(c) analyzes the CGS with a model checker for property conformance.

A flow diagram for the relevant part of the software is presented in Figure 5.1. The C_{all} is provided in the Promela language which is the input language for SPIN. The recommended task is given as a sequence of human operator actions using C_{all} vocabulary. The protection envelope or safety property is specified using LTL. We have described the interesting mechanisms of the tool like C_{all} creation from Promela input, CGS creation for a task description, CGS encoding into Promela for analysis by SPIN in Section 4.8. After a task specification model has been created, we first determine whether it is physically possible to be executed. Physical impossibility of execution is indicated by the task specification requiring an action to be performed by the operator at a state where it is not contained in the e function of that operator in that state in the CGS. A task variation that is not physically possible is inherently safe as the system is capable of barring the human operator from executing the task. Only feasible task specifications are analyzed for being protected or safe.

5.4 AIDC Scenario Revisited

Let us consider a simple AIDC scenario with two patients `pat1` and `pat2` and two medical devices `dev1` and `dev2`. There is only one nurse and one medical mediator. The nurse is supposed to take a reading from `pat1` using `dev1`. Later on she is supposed to take a reading from `pat2` using `dev2`. Simple recommended task descriptions for the nurse for these two procedures are: (1) press start, scan `pat1` ID, scan `dev1` ID, verify IDs displayed, take reading, verify captured data for storage in the back end database; and (2) press start, scan `pat2` ID, scan `dev2` ID, verify IDs displayed, take reading, verify captured data for storage in the back end EHR database. The protection envelope for the first recommended task is:
If the nurse verifies `pat1ID` and `dev1ID` then previously she had scanned only `pat1ID` and `dev1ID` and *if the nurse verifies `pat1ID`, `dev1ID` and data `d` then the data `d` WAS captured from `pat1` using `dev1`*.

Now let us analyze the first recommended task description for the nurse. We discuss the erroneous action patterns that have been included in the tool. The erroneous actions that are not mentioned here have not yet been implemented.

- Repetition: One time repetition of Press Start is feasible and safe. This might seem surprising, but this is only feasible due to the fact that we allow τ actions for every player in every state. Thus the mobile mediator is allowed to not respond to the pressing of the Start button by the nurse. We allow the silent action τ to included as a possible action for each agent in a CGS. We allow silent transitions so that we can model situations where the human operator does not respond immediately to computer system actions and takes some time to start performing an action. On the other hand, although a human operator may believe that he has performed an action requiring concurrent response from the computer system, but the computer system may not have registered the fact that an action has been performed by the human operator. Repetition of scanning the ID tag of the patient and the device are also feasible. This is a very important result, because scanning a tag multiple times can be a very common phenomenon while using the AIDC system. This assures that the nurses will not need to restart the procedure if they have inadvertently scanned the same ID multiple times.

- Reversal: Quite unsurprisingly, we find that the Press Start button action cannot be permuted with any other action. If a nurse wants to perform AIDC using the medical mediator she needs to press the start button as the first step of the entire procedure. The most significant finding for reversal is that scanning of the ID for the patient and the device can be swapped. Hence nurses will not need to restart the whole procedure if she has scanned the RFID tag of the device before scanning the RFID tag of the patient.

- Omission: Similar to the findings discussed above omission of the Press start button action is infeasible. Also the nurse needs to scan the IDs of the patient and device in some order for the process to progress. Hence omitting these actions will result in the entire process getting stalled.

- Delay: Since we do not perform timed analysis we are not capable of analyzing the effect of delaying on part of the operators. However, since we insist that the silent action τ be an enabled action for each player in each state in the CGSs it provides a mechanism for modeling delayed action. This is evident from the analysis results from the repetition of actions step. Although automated systems are supposed to be capable of almost immediate response, in the case that the system *is* late in responding, the operator might have gotten impatient and pressed a button twice. In future we will undertake inserting the silent τ action an arbitrary number of times into the recommended task specification to incorporate delayed human action analysis in our framework.

- Premature action/ Jump forward/ Jump backward: Instead of considering these patterns of execution separately, we inserted each action from the the recommended task specification in front of each action in the same recommended task specification. In the absence of timing in our models, the best approximation we could achieve of a subtask getting performed too early was its getting executed before another subtask. We obtained that most of the actions like scanning the ID tags, obtaining a reading from a device cannot be performed ahead of the Press start button action. The ID tags of the devices can however be scanned before the ID tags of the patients are scanned. This result is very similar to the result obtained by reversing the order of the actions.

- Intrusion: For this analysis, we inserted subtasks from the recommended task specification for AIDC from the second patient into the recommended task specification for the first patient. We did this for two different implementations of the AIDC scenario. During the initial stage of the AIDC project, the system under development was somewhat rigid in the sense that once the ID tag of a patient was scanned only a device ID tag could be scanned in the next step. Then the design was made more flexible by allowing the same ID tag of a patient or a device to be scanned multiple times. Ideally, an RFID tag scanner cannot distinguish among the tags belonging to different entities. The mobile mediator can determine whether the tag is a patient tag or a device tag after it has consulted with the back-end database. Hence the most flexible form of the system should allow all RFID tags to be scanned in the ID scanning steps. Then later on, when the nurse is asked to verify that the scanned IDs are acceptable, the nurse gets a chance to correct any errors. We analyzed the last two models. At this point we intend to focus on an unusual finding on our part. The finding occurred in the second version of the AIDC model (first version analyzed by us). In this model once the tag of a patient has been scanned, only that tag can be scanned multiple other times. Then the ID tag of a device needs to be scanned. Now, our findings surprisingly contained the result: "`insertion of scanPat2ID after scanPat1ID is physically` `possible`"(may not be safe). Although this may seem to be an erroneous judgment, the reasoning that leads to this is as follows:

- Nurse presses the start button.
- Nurse scans the ID tag of the first patient. The combination of the RFID tag scanner and the mobile mediator is slow enough (modeled using the silent τ action) that the nurse decides that she was not able to scan the ID tag of the patient successfully.
- Nurse now scans the ID tag of another patient because she wants to proceed with the AIDC procedure of the second patient.

We had decided that the τ action be available to each entity at each step to enable modeling delayed response from them. This result alerts us that if it is possible for the automated systems to not respond to the operator actions in an immediate manner, the operators may get confused and decide to wait, re-perform the step or even worse insert a step from a different task specification. A modeling of the scenario where computer system components are not allowed to stay idle through the τ actions would not have uncovered this issue. The analysis results are available on the Tutela website.

5.4.1 Protection Envelope Suggestion

The methodology presented can be used alternatively for checking the robustness of a safety property. Then the analysis will indicate which erroneous human behaviors are safe for the system and which ones are not. The collection of erroneous human behaviors that are not hazardous for a system can constitute a protection envelope for the system. Let us consider the AIDC scenario again. The safety property for the AIDC scenario is very similar to the protection envelope property presented earlier.

If the EHR stores (`nurseID`*,* `mobileMediatorID`*,* `pat1ID`*,*`dev1ID`*,* `reading, r`*) then previously she had scanned only* `pat1ID` *and* `dev1ID` *and the reading* `r` *WAS captured from* `pat1` *using* `dev1`*.*
We analyzed the AIDC system with respect to this safety property under erroneous variations of the recommended nurse task description. Let us consider some parts of the analysis result:

- Repeated scanning of the RFID tag of a patient is safe.
- Repeated scanning of the RFID tag of a device is safe.
- The order of scanning the RFID tags of a patient and a device can be reversed.
- ID validation step cannot be performed before the RFID tags have been scanned.

From these results, one can observe that as long as the same patient and device is scanned it does not matter how many times we scan them. It also does not matter whether the device or the patient was scanned first. Thus as long as the nurse validates the correct patient and the correct device identification information, the nurse behavior is safe. This observation can be translated into:

$$\Box(\neg \mathtt{patdevIDOked(i,j)}\ \mathcal{U}\ (\ \mathtt{patIDScanned(i)} \wedge\ \mathtt{devIDScanned(j)}))$$

This property states that the nurse does not validate the identities of the *i*-th patient and *j*-th device until she has actually scanned them. This can be a protection envelope for the nurse. This protection envelope contains the recommended task description of first scanning the RFID tag of the patient and then scanning the RFID tag of the device. It also includes the behavior where the device is identified before the patient. It also includes multiple scanning of the RFID tags of the patients and the devices.

5.5 Summary

In this chapter we presented how to test the robustness of a protection envelope against typical human errors. We also show how these analysis can help formulate a protection envelope itself if no protection envelope is present.

Chapter 6

Modeling Human Behavior with Incomplete Information

In this chapter we provide a more fine grained formalism for modeling and reasoning about human behaviors.

6.1 Introduction

A model for human behaviors exhibited in conjunction with a system should be able to encode the actions they will perform at different states. We had decided on CGSs as the model for human behavior as they have agent identities, agent actions and the possibility of expressing properties where a particular group of agents can provide a guarantee using ATL or variants of ATL like ATL*. Thus CGSs provided us a mechanism that allowed for procedural encoding of recommended behaviors, declarative encoding of safe, unsafe, effective and protected behaviors. ATL have not been utilized as of yet due to several reasons. Syntactic restrictions of ATL rendered them somewhat undesirable for expressing protection envelope properties. Let us consider the Self-scan checkout scenario. A protection envelope for the customer is: $\langle\langle \mathit{Customer} \rangle\rangle \Box(\mathtt{outofStore(i)} \rightarrow \blacklozenge\mathtt{paidForitem(i)})$. This property can only be expressed using ATL* (with necessary conversion for past time operators) not ATL. We focused on a specific subset of ATL*: LTL. The protection envelopes that we have considered were invariants of the form $\Box\varphi$ where φ is an LTL formula. LTL formulae served our purposes and had the advantage of having a wide range of model checking softwares being available. We now refocus on the CGSs, ATL and ATL* to be able to present a more precise modeling formalism and analysis technique for human behavior.

6.1.1 Contributions

Unlike the earlier chapters the contributions of this chapter are more theoretic in nature.

- We first present the shortcomings we found of modeling human behavior through the enabled action function of CGSs (i) deterministic transition system, (ii) being incapable of modeling history dependent human behavior and (iii) absence of information hiding among agents in a CGS. Then we present an enhanced version of CGSs namely iNCGSs capable of overcoming the issues of deterministic transition system and absence of information hiding with the usual form of CGS.

- We present ATL semantics using iNCGS. We provide a model checking algorithm for ATL semantics using iNCGS.
- We show how to preform semantics preserving transformation of incomplete information turn based game structures into iNCGS.
- We also provide ATL* semantics using iNCGS.
- We show how human behavior can be modeled using player strategies to allow modeling history sensitive human behavior. We show how to determine if a strategy satisfies an ATL formula to provide a better formal technique of determining when a human behavior conforms to a guarantee.
- We present nondeterministic strategies and present strategic semantics of ATL*.
- We also show how other work on incomplete information multi-agent structures can be related to iNCGS.

6.1.2 Organization

We first present an example scenario in Section 6.2 where human operators play an important role in the (un)safe executions of the relevant computerized systems. Then we show how the formalism chosen in Section 4.5 can model the human operator behaviors in Section 6.3. Then we present the shortcomings of that formalism in modeling human operator-intensive systems in Section 6.4. Then we present the enhanced model iNCGS and ATL and ATL* semantics using iNCGS in Section 6.5. We show how iNCGS can be related to other multi-agent game structures in Section 6.6. We present some variations of iNCGS in Section 6.7. We present how strategies can model human behaviors in Section 6.8. We present strategic semantics of ATL and ATL* in Section 6.9. We summarize our results in Section 6.10.

6.2 Example Scenario: 2001 Japan Airline Mid-air Incident

Two Japan Airlines flights nearly collided in mid-air on Wednesday, January 31, 2001 due to human error [117, 102]. Japan Airlines Flight 907 registered as JA8904 was going south-west from Tokyo International Airport (Haneda Airport) in Ota, Tokyo, Japan to Naha International Airport in Naha, Okinawa, Japan. We will refer to it as aircraft A. Japan Airlines Flight 958 registered as JA8546 bound from Gimhae International Airport in Busan, South Korea was going east to Narita International Airport in Narita, Chiba Prefecture, Japan. We will refer to it as aircraft B. Aircraft A was ascending to 39,000 feet while making a left turn above water when a conflict was detected at the Tokyo Area Control Center (TACC) because aircraft B was flying at a similar flight level. Aircraft B was cruising at 37,000 feet towards aircraft A. The Traffic Alert and Collision Avoidance System (TCAS) in aircraft A issued a Resolution Advisory (RA) for *ascending higher* to avoid a possible collision with a dangerously nearby aircraft: the aircraft B. The TCAS in aircraft B had issued an advisory for descending to avoid any possible collision with the ascending aircraft A. The trainee operator at TACC was upset and in that state of mind issued *descend* request to the *wrong* aircraft: aircraft A. Aircraft A actually needed to ascend to ensure collision avoidance.

An interesting scenario developed at this point. In spite of the TCAS advisory, the aircraft A pilot decided to follow the TACC instructions to descend. On the other hand, the TACC detected that aircraft B was not descending and requested it to change heading to avoid collision. This instruction to aircraft B was lost and aircraft B had only its TCAS advisory to descend at its disposal. Aircraft B pilot decided to follow the advisory. Thus both aircrafts ended up descending while approaching each other. They ended up being able to see each other and a collision was avoided because at the final moment the pilot of aircraft A drastically reduced their altitude. No passengers in aircraft B was injured. Refreshments were being served on board aircraft A and several passengers were not wearing seat belts, hence there were some injuries among the passengers there. The entire episode lasted less than two minutes. During the entire process the TCAS for each aircraft kept issuing warnings and advisories. These warnings were correct and appropriate. The TACC operators kept issuing directions too. However in the heat of the moment they lost their nerve, issued wrong direction to wrong aircraft and even issued a direction to an aircraft with a similar number which was not even flying at that time. The aircraft B pilots received partial information and acted accordingly. The aircraft A pilots received information from both their aircraft and the control towers but could not take the correct step. On the other hand the control tower operators were not aware that the TCAS of the two aircrafts had issued warnings and advisories. This information would have helped them understand their own mistake and the real situation in the airspace. In this scenario human error caused the incident and human judgment avoided it. Let us now formally model this scenario to analyze human operator actions.

6.3 Human Behavior Model

The chosen formalism for implementing the framework presented in Section 4.5 was CGS. Let us consider the definition of CGS again. CGS is an 8-tuple $\langle P, Q, q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ where the components are as follows: P is a nonempty finite set of players. Q is a finite set of states and $q_0 \in Q$ is the initial state. Σ is a finite nonempty set of actions for the players where the silent action $\tau \in \Sigma$. Π is a finite set of atomic propositions. $\pi : Q \to 2^{\Pi}$ is a function that gives the set of propositional atoms that hold of a given state. $e : P \times Q \to 2^{\Sigma}$, gives actions that are enabled for any given player in any given state. The restrictions on the enabled actions function e are: (i) $\forall p \in P. \forall q \in Q. |e_p(q)| > 0$,

(ii) $\forall p \in P. \forall q \in Q. \tau \in e_p(q)$. These two restrictions state that at each state each player should be able to perform at least one action. And also each player should be given the chance to remain idle at a state if he so chooses. Given a vector of action choices $\bar{a}$, we denote the action chosen by player p in that vector by $\bar{a}|_p$ and the actions chosen by a set of players $\mathcal{P}$ as $\bar{a}|_{\mathcal{P}}$. The transition function $\delta : Q \times (e_{p_1}(q) \times e_{p_2}(q) \times \ldots \times e_{p_{|P|}}(q)) \rightarrow Q$ defines the state transitions. That is, given a state q, and a vector of actions $\bar{a} \in \Sigma^P$ where $\bar{a}|_p \in e_p(q)$ for each p, the state determined by $\delta(q, \bar{a})$ is the next state. δ has the restriction: $\forall q \in Q.\ q \in \delta(q, \bar{\tau})$. τ action and $\bar{\tau}$ transitions allow human operators to catch up with automated systems. Players can have strategies in a CGS for choosing actions at states. A strategy for a player $S_p : Q^+ \rightarrow \Sigma$ an action to p for every finite prefix of computation.

CGSs were found suitable because they allow (i) identifying the important players in a scenario, (ii) identifying the action vocabularies of the players, (iii) restricting the behavior of the players by restricting the actions available to the players in each state. We modeled human behavior through the enabled action function e of a CGS. Human behavior is how they respond to a situation $\rightarrow$ the actions they may choose from in a state $\rightarrow$ the enabled action function e. CGSs allow capturing the transition of a system through player actions. Also, the logic imposed on CGSs is ATL. ATL allows one to express the formulae of the form $\langle\!\langle A \rangle\!\rangle \varphi$ stating that the group of players in A can guarantee φ irrespective of the behavior of the rest of the players. The syntax of ATL over a set of propositions Π is as follows:

$$\varphi = p \in \Pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\!\langle A \rangle\!\rangle \bigcirc \xi \mid \langle\!\langle A \rangle\!\rangle \Box \xi \mid \langle\!\langle A \rangle\!\rangle \xi_1\ \mathcal{U}\ \xi_2$$

Here $\bigcirc$, $\Box$ and $\mathcal{U}$ are next step, always and until operator like Linear Temporal Logic (LTL). An ATL formula of the form $\langle\!\langle A \rangle\!\rangle \varphi$ states that there exists a strategy for each player in the player coalition A such that irrespective of any actions taken by the rest of the players, the players in A have a way of guaranteeing φ. This formula essentially states that the players in the group A *may* guarantee φ. However, we are more interested in *must* guarantees. A human operator *must* maintain the guarantee expressed by his protection envelope. Thus we are driven towards ATL formulae of the form $[\![A]\!]\varphi$. The intuitive meaning of $[\![A]\!]\varphi$ is that the players in A cannot avoid φ, they cannot cooperate to make it be false. More precisely, $[\![A]\!]\Box\varphi$ is defined to be $\neg \langle\!\langle A \rangle\!\rangle \Diamond\neg\varphi$, $[\![A]\!] \bigcirc \varphi$ is defined to be $\neg \langle\!\langle A \rangle\!\rangle \bigcirc\neg\varphi$. $[\![A]\!]\varphi_1\mathcal{U}\varphi_2$ is defined similarly. $[\![A]\!]\varphi$ can also be interpreted as: no matter what actions are performed by the players in A, the rest of the players can find a strategy to ensure φ. Given these interpretations, we choose protection envelopes to be of the form $[\![\texttt{Human Operator}]\!]\Box\varphi$ i.e. the human operator cannot avoid behaving according to the protection envelope property. Here presumably, the rest of the players in the CGS is just the computer system. A protection envelope is a guarantee that the human operator will provide, hence the only way the computer system can guarantee that no matter what actions the human operator performs the human operator behavior will maintain protection property is if the human operator actually never performs any unsafe action. Thus this type of protection envelope will hold only in those CGSs where possible protection envelope violating human operator actions are disabled in each state i.e. the human operator does not take any "bad" step. Thus every step performed by human operator is a good step and hence the protection envelope guarantee of the human operator will hold. Now, just as we said earlier, we prefer protection envelopes expressed using ATL*. The syntax of ATL* over a set of propositions Π is as follows:

$$\varphi = \nu \in \Pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\langle A \rangle\rangle \psi$$
$$\psi = \varphi \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \bigcirc \psi \mid \psi_1 \,\mathcal{U}\, \psi_2$$

Here $\bigcirc$ and $\mathcal{U}$ are next step and until operator like Linear Temporal Logic(LTL). ATL* includes LTL where $\langle\langle\ \rangle\rangle\psi$ stands for LTL formulae. $\langle\langle\ \rangle\rangle\psi$ states that no matter what actions all the players perform the property ψ will hold.

6.3.1 Model of the Japan Airlines Incident using CGS

Let us model the Japan Airlines incident using CGSs. The players in the CGS are: `AircraftA`, `AircraftB` (the control panel portion of these aircrafts); `AirCraftAPilot`, `AircraftBPilot` and `ATC` (humans controlling the TACC). The action vocabulary for the pilots can be:

`flyStraight`, `maneuverAscend`, `maneuver descend`, `changeHeading` etc.

The aircrafts' possible actions are:

`calculateSpeed`, `detectCollision`, `adviseAscent`, `adviseDescent` etc.

The `ATC` vocabulary can be comprised of:
`monitorAirSpace`, `requestAscend`, `requestDescend`, `requestChangeHeading`, `permitToLand` etc.

We will focus on interactions among aircrafts flying through a specific segment of air space and the air space controllers. We will not consider aircraft landing and takeoff procedures at this time. We want to ensure that given a particular airspace no two aircrafts are in the same air space zone at the same altitude at a particular point in time. Very abstractly this can be formulated as

$\langle\langle$ `ATC` $\rangle\rangle \square(\neg($`AirCraftAInAirSpaceWindow1` $\wedge$ `AirCraftBInAirSpace Window1`$))$. The protection envelope of the pilots need to state that if the pilot performs ascend or descend then there were either an `ATC` directive or TCAS directive to do so. For example:

$[\![$`AirCraftAPilot`$]\!]\square($`AircraftAascending`$\rightarrow$ $(\blacklozenge$ `ATCreqAscendtoA`$) \vee (\blacklozenge$ `AscendAdvisoryA`$))$ The CGS corresponding to this scenario (partial) is given in Figure 6.1. In the figure, the player action combinations are placed next to the states. The action of the `ATC` is listed first, then `AirCraftAPilot`, `AircraftBPilot`, `AircraftA` and `AircraftB`'s actions are listed. We annotate the states with some relevant state propositions enclosed within square brackets.

Let us consider a state q. In this state the `ATC` detects a conflict. Then the system moves on to state q_1. Then `AircraftB` detects a possible collision ahead and the system moves to q_2.

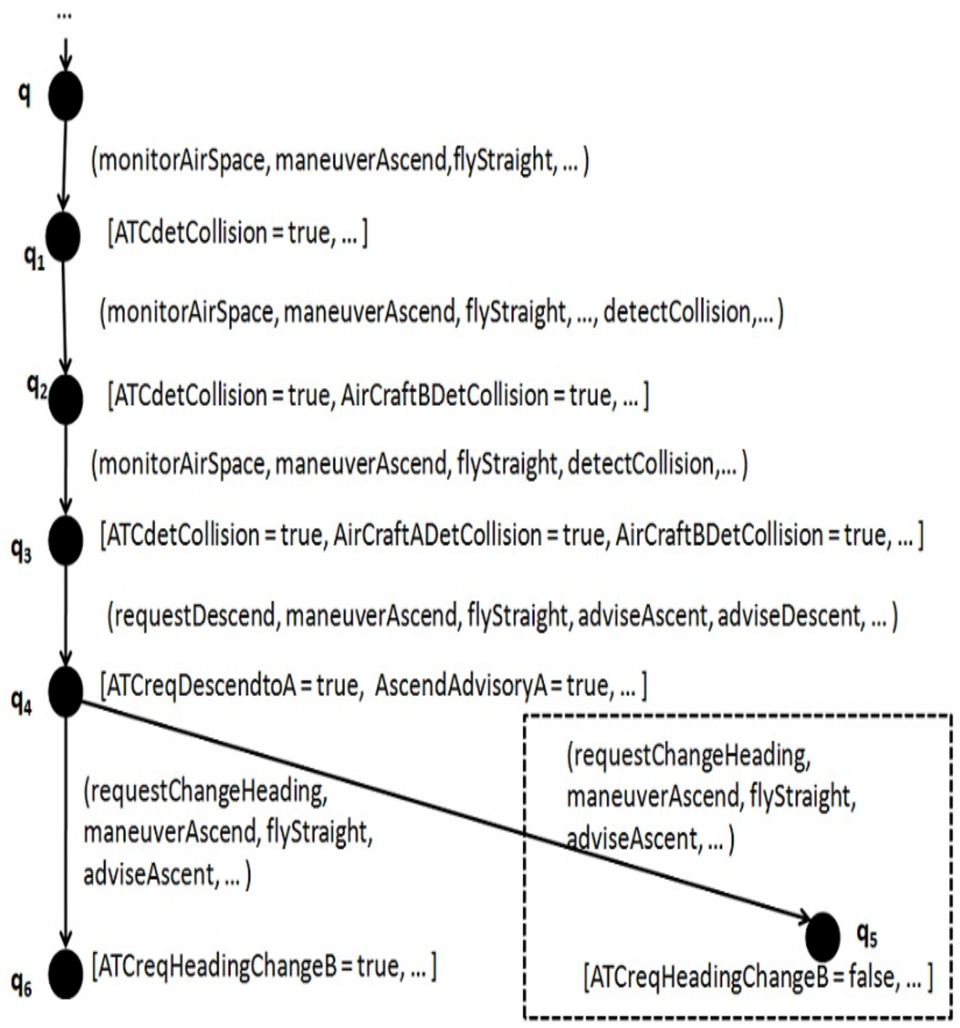

...
q
(monitorAirSpace, maneuverAscend,flyStraight, ...)
q1
[ATCdetCollision = true, ...]
(monitorAirSpace, maneuverAscend, flyStraight, ..., detectCollision,...)
q2
[ATCdetCollision = true, AirCraftBDetCollision = true, ...]
(monitorAirSpace, maneuverAscend, flyStraight, detectCollision,...)
q3
[ATCdetCollision = true, AirCraftADetCollision = true, AirCraftBDetCollision = true, ...]
(requestDescend, maneuverAscend, flyStraight, adviseAscent, adviseDescent, ...)
q4
[ATCreqDescendtoA = true, AscendAdvisoryA = true, ...]
(requestChangeHeading, maneuverAscend, flyStraight, adviseAscent, ...)
(requestChangeHeading, maneuverAscend, flyStraight, adviseAscent, ...)
q5
[ATCreqHeadingChangeB = false, ...]
q6
[ATCreqHeadingChangeB = true, ...]

Figure 6.1: Japan Airlines incident in CGS

Then AircraftA detects a possible collision leading to state q_3. Now the ATC erroneously requests AircraftA to lower altitude. Thus in this state the enabled actions for the ATC should include the desired request of descend to AircraftB in addition to all possible erroneous requests. Then AircraftB detects collision again and issues resolution advisory suggesting descend. Then AircraftA provides a resolution advisory suggesting ascend to higher altitude to avoid collision. Then the ATC detects that despite the warning (to the wrong aircraft) AircraftB is not descending to avoid collision. Then ATC requests AircraftB to change heading to avoid collision. However, this communication is lost due to error and no action is taken by AircraftBPilot. However, given a combination of player actions, a CGS can only transition to one particular state. Hence with the action vector comprising of sending a heading change request by the ATC, we can either have the transition from q_4 to q_5 or from q_4 to q_6 in the CGS.

6.4 Problems with Modeling with the Previous Approach

Several issues are highlighted by the CGS model as presented in the last subsection. Let us consider them one by one. First, the CGS transition system is deterministic. When an ATC sends an instruction to an aircraft, the communication may or may not reach that aircraft. Instructions issued to AircraftB did not reach AircraftB. Thus with the same combination of player action choices, the CGS should be capable of choosing among a set of states to transition to. In the scenario we are considering, the TACC are not made aware of the warnings issued by the TCAS of the two aircrafts. But in the CGS model the ATC can detect when an advisory has been issued for either plane. The status of each and every proposition and variable are available to each player in a CGS. Pilots of either aircraft should not be aware of what signals are being issued by the other plane's TCAS to their corresponding pilots. In this scenario, the pilots can see all actions of the other aircraft.

In order to avoid a collision two aircrafts approaching each other should not both ascend or descend. In the least they should change their headings in opposite directions. In the CGS, we observe that AirCraftAPilot can see that AircraftBPilot has received no warning from the ATC, the AircraftB has issued a descend advisory, his AircraftA has issued an ascend advisory and a descend advisory has reached him from the ATC. From all these information he can deduce that the only option for AircraftBPilot is to descend. Hence he himself should not descend but get confirmation from ATC or ascend according to TCAS advisory. Such a strategy can be formulated in the CGS which is actually an unfaithful modeling of the scenario. To faithfully capture the scenario, we need to be able to hide information from players.

The notion of inadequacy of information cannot be modeled by CGS's in their current form. The players are not "module" like with an incomplete view of the states of the world. Every player has access to information that should have been visible to only a select set of players.

Alur *et al* presented Turn Based Synchronous Game Structure with incomplete information (iTSGS) in [9] to be able to model information hiding. We discuss iTSGS below.

6.4.1 iTSGS: incomplete Information Turn Based Synchronous Game Structure

A Turn Based Synchronous Game Structure with incomplete information is a tuple ($S = \langle P, Q, \Pi, \pi, \sigma, R\rangle, \mathcal{P}$) where S is a Turn Based Synchronous Game Structure and $\mathcal{P}$ provides the propositions that each players can observe. Here P is a set of players, Q is a set of states, Π is a set of atomic propositions, $\pi : Q \to 2^{\Pi}$ defines which propositions are true in which state. $\sigma : Q \to P$ defines which player gets to act in which state. $R \subseteq Q \times Q$ is a total transition relation. They model incomplete information through the observability vector $\mathcal{P} : P \to 2^{\Pi}$. It defines which propositions can be observed by each player. A player cannot distinguish between two states if he cannot observe the only proposition distinguishing them. They use $\pi_a(q)$ to denote the propositions observable to player a in state q i.e. $\pi_a(q) = \pi(q) \cap \mathcal{P}_a$. This notation can be easily extended to sequences of states: $\pi_p(q_0, q_1, q_2, \ldots) = \pi_p(q_0), \pi_p(q_1), \pi_p(q_2), \ldots$. Any subset of π_a for a player a is called an a-view. And the set of all possible a-views is denoted by V_a. A strategy for a player p is a function $f_p : V_a^+ \to V_a$. Thus given a computation prefix, the strategy suggests which states the system can go to when it is p's turn to execute.

However, we found iTSGS too simplistic to be a sufficient model for us. In an iTSGS, the players do not have a specific set of actions to choose from. The vocabulary of players is implicit. The transition system is represented with a relation and hence non-deterministic. But, the player actions do not implicitly influence the transitions whereas our primary focus is on the player actions. Human operator behaviors are expressed through the actions they take at different states. Hence, having explicit action vocabulary and enabled action information in states is important to us. Also we show later that the strategies of CGSs are more suitable to modeling human behavior. Given a computational history, a CGS strategy for a player p indicates what action p should take. In the example scenario, the strategy of `AirCraftAPilot` is to follow the `ATC` instruction after receiving instructions both from the `ATC` and his own aircraft. Thus we present an extension of CGS namely Nondeterministic CGS with imperfect recall (iNCGS) to more naturally capture human operator behaviors.

6.5 Nondeterministic CGS with Incomplete Information

6.5.1 iNCGS

We present an extension to CGSs having more generalized notion of imperfect observational capability for the player along with non-deterministic transition system. We call this extension iNCGS for *i*ncomplete information *N*ondeterministic CGS. An iNCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta\rangle$ is a 10-tuple. P, Q , $Q_0 \subseteq Q$, Σ, Π and π and e are as defined in Section 6.3. The set of restrictions mentioned about these constructs for CGSs carry over to the iNCGSs. The transition function δ is the first modified component: $\delta : Q \times e_{p_1}(q) \times e_{p_2}(q) \times \ldots \times e_{p_{|P|}}(q) \rightarrow 2^Q$. That is, given a state q, and a vector of actions $\bar{a} \in \Sigma^P$ where $\bar{a}|_p \in e_p(q)$ for each p, the states in $\delta(q, \bar{a})$ are the possible next states. δ still has the restriction: $\forall q \in Q.\ q \in \delta(q, \bar{\tau})$. τ action and $\bar{\tau}$ transitions allow human operators to catch up with automated systems. A *q-computation* r of an iNCGS C is an infinite sequence of states where $r_0 = q$ and for each $i \geq 0$, there is a vector $a_i \in e_{p_1}(r_i) \times e_{p_2}(r_i) \times \ldots \times e_{p_{|P|}}(r_i)$ of actions such that $r_{i+1} \in \delta(r_i, a_i)$.

iNCGSs model incomplete information through state and trace equivalence classes. $\dot{\sim}$ is a set of equivalence classes on states. We will denote the equivalent classes for each player p by $\dot{\sim}_p$ where $\dot{\sim}_p : 2^{Q \times Q}$. A player p cannot distinguish between two states q_1 and q_2 where $q_1 \dot{\sim}_p q_2$. We will denote the equivalence class of a state q with respect to the point of view of player p by $[q]_p$. We require that e conforms to $\dot{\sim}$ i.e. for a player $p \in P$ and a state $q \in Q$, $\forall q' \dot{\sim} q.\ e_p(q) = e_p(q')$. In other words, if a player cannot distinguish between two states then he should have the same action choices in both of them.

$\ddot{\sim}$ is a set of equivalence classes on sequences of states. $\ddot{\sim}_p$ denotes the "history" equivalence classes for p. Given two finite sequences of states λ_1 and λ_2, if $\lambda_1 \ddot{\sim}_p \lambda_2$ then p cannot distinguish between them. We require that $\ddot{\sim}_p$ conform to $\dot{\sim}_p$: (i) $\dot{\sim}_p \subset \ddot{\sim}_p$ and (ii) if $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \in Q^+$ and $\lambda_1 \ddot{\sim}_p \lambda_2$ and $\lambda_1 \ddot{\sim}_p \lambda_2$ then $\lambda_1 \cdot \lambda_3 \ddot{\sim}_p \lambda_2 \cdot \lambda_4$.

Our notion of trace equivalence is more generalized in that we can define trace equivalence relations in whatever way we please. For example: we can define traces to be stuttering equivalent or inequivalent under stuttering. For this work, we impose the following restrictions on the trace equivalence classes: for a player $a \in P$, $\lambda_1 \ddot{\sim}_a \lambda_2 \Leftarrow |\lambda_1| = |\lambda_2|$ and $\forall 0 \leq i \leq \lambda_1.\ \lambda_{1,i} \dot{\sim}_a \lambda_{2,i}$. Let us extend the notions of state equivalences and trace equivalences from the point of view of a single player to the point of view of a set of players.

- $\dot{\sim}_{\mathcal{P}}$: $q_1 \dot{\sim}_{\mathcal{P}}\ q_2 = \forall p \in \mathcal{P}.\ q_1 \dot{\sim}_p q_2$
- $\ddot{\sim}_{\mathcal{P}}$: $\lambda_1 \ddot{\sim}_{\mathcal{P}} \lambda_2 = \forall p \in \mathcal{P}.\ \lambda_1 \ddot{\sim}_p\ \lambda_2$
- $[q]_{\mathcal{P}} = \{q' | \, q \dot{\sim}_{\mathcal{P}}\ q'\}$
- $[\lambda]_{\mathcal{P}} = \{\lambda' | \, \lambda \ddot{\sim}_{\mathcal{P}} \lambda'\}$

A strategy for a player p is a function $S_p : Q^+ \rightarrow \Sigma$ that given a history of visited states, S_p suggests the next action to take. A strategy S_p has to conform to equivalence classes of states: $\forall \lambda_1, \lambda_2 \in Q^+$. if $\lambda_1 \ddot{\sim}_p \lambda_2$ then $S_p(\lambda_1) = S_p(\lambda_2)$ i.e. strategy should make the same suggestion to a player p in states that are indistinguishable to p. The outcome O of following a strategy $S_{\mathcal{P}}$ by a group of players $\mathcal{P}$ in a state q is defined as $O(q, \mathcal{P}, S_{\mathcal{P}}) = \lambda$ where $\lambda_0 = q$ and for each $i \geq 0.\ \lambda_{i+1} \in \delta(q, \bar{v})$ where $\bar{v}|_{\mathcal{P}} = S_{\mathcal{P}}(\lambda_0^i)$.

6.5.2 Example Scenario using iNCGS

Now we introduce the nondeterminism and information hiding in the CGS as presented in Figure 6.1. Firstly, in iNCGSs, we can have nondeterministic transitions. Hence, we can have both the q_4 to q_5 and q_4 to q_6 transitions. The `ATC` should not be aware of the warnings and advisories being issued by the TCAS systems of the aircrafts. Hence, to the `ATC`, the states q_1, q_2 and q_3 should be the same. So, $[q_1]_{ATC} = \{q_1, q_2, q_3\}$. The `ATC` can not detect if a heading change request by him has reached the `AircraftBPilot` or not. Hence, he cannot detect the difference between states q_5 and q_6. Thus we have, $[q_4]_{ATC} = \{q_5, q_6\}$. For the the first pilot, he should not be aware of the situation that another aircraft has detected a collision. Hence for `pilot1`, we can set $[q_1]_{\texttt{pilot1}} = \{q_1, q_2\}$. Similarly, for `AircraftBPilot` we can have: $[q_2]_{\texttt{pilot2}} = \{q_2, q_3\}$.

6.5.3 Semantics of ATL with iNCGS

Given an NCGS C, a sequence of states λ and an ATL formula φ, we present the semantics of ATL using iNCGS:

- $q \models_I v \in \Pi$ if $v \in \pi(q)$.
- $q \models_I \neg\varphi$ if $q \not\models \varphi$
- $q \models_I \varphi_1 \vee \varphi_2$ if $q \models_I \varphi_1$ or $q \models_I \varphi_2$
- $q \models_I \langle\!\langle \mathcal{P} \rangle\!\rangle \bigcirc \psi$ if there exists a set of strategies $S_{\mathcal{P}}$ such that $\forall q' \dot{\sim}_{\mathcal{P}} q.\ \forall \lambda \in O(q', \mathcal{P}, S_{\mathcal{P}}).$ $\forall \lambda' \ddot{\sim}_{\mathcal{P}} \lambda. \lambda'_1 \models \psi$
- $q \models_I \langle\!\langle \mathcal{P} \rangle\!\rangle \Box\psi$ if there exists a set of strategies $S_{\mathcal{P}}$ such that $\forall q' \dot{\sim}_{\mathcal{P}} q.\ \forall \lambda \in O(q', \mathcal{P}, S_{\mathcal{P}}).$ $\forall \lambda' \ddot{\sim}_{\mathcal{P}} \lambda. \forall 1 \leq i \leq \infty. \lambda'_i \models \psi$
- $q \models_I \langle\!\langle \mathcal{P} \rangle\!\rangle \psi_1 \,\mathcal{U}\, \psi_2$ if there exists a set of strategies $S_{\mathcal{P}}$ such that $\forall q' \dot{\sim}_{\mathcal{P}} q.\ \forall \lambda \in O(q', \mathcal{P}, S_{\mathcal{P}}).$ $\forall \lambda' \ddot{\sim}_{\mathcal{P}} \lambda. \exists 1 \leq j \leq \infty. \lambda'_j \models \psi_2 \wedge \forall 1 \leq i < j. \lambda'_i \models \psi$

6.5.4 ATL Symbolic Model Checking over iNCGS Models

The *model checking problem* for a given iNCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta \rangle$ and and ATL formula φ asks for the set of states that satisfies φ. We reuse the symbolic model checking algorithm for solving the model checking problem for ATL formula and CGSs with complete information as given in [9] with some changes. The algorithm presented in [9] is defined for a deterministic CGS with deterministic transition system. One can translate a nondeterministic transition system structure like iNCGS into a deterministic one by introducing an oracle player who will decide among the possible outcome states for every transition with multiple possible outcome states. This idea is very similar to the Harsanyi transformation [53] in game theory.

The model checking problem for an ATL formula φ on an iNCGS I asks for the states $q \in Q$ such that $q \models \varphi$. Given an iNCGS I, we denote the desired set of states as $\Upsilon(\varphi)$. The ATL symbolic model checking algorithm presented in [9] is modified for our setting in Algorithm 1 to handle state equivalence classes. The auxiliary functions used by the algorithm are as follows:

- The function *Sub*, when given an ATL formula φ, returns a queue of syntactic subformulas of φ such that if φ_1 is a subformula of φ and φ_2 is a subformula of φ_1, then φ_2 precedes φ_1 in the queue $Sub(\varphi)$.
- The function *Reg*, when given a proposition $p \in \Pi$, returns the set of states in Q that satisfy p.
- The function *Pre*, when given a set $A \subseteq P$ of players and a set $\rho \subseteq Q$ of states, returns the set of states q such that from q or any other state equivalent to q from the point of view of players in A can cooperate and enforce the next state to lie in ρ. Formally $Pre(A, \rho) = \{q \mid \exists \bar{v} \in \prod_{p \in A} e_p(q).\ \forall \bar{v'} \in \prod_{p \in P \setminus A}.\ \forall q' \in [q]_A.\ \delta(q', \bar{v} \circ \bar{v'}) \in \rho\}$.

The $\mathcal{E}$ function takes a set of states ρ and a set of players A and returns another set of states defined as follows: for each $\mathcal{E}(\rho, A) = \{q \mid q \in [q']_A$ and $q' \in \rho\}$ i.e. $\mathcal{E}$ populates the set of states ρ with each and every state that is observationally equivalent from the point of view of the players in A to the states already in ρ. Thus if there is a state q from which the players in A can guarantee to make the iNCGS to land in q' then from each and every state $q'' \in [q]_A$, the players in A can force the structure to land in q' or another state $[q']_A$. The only change in the setting is due to the state equivalence relations. Here a player a will have a strategy to ensure φ in the next step from a state q iff it can ensure φ from each state q' that is equivalent to it. This affirms that the behavior of a player p is uniform across all states that are indistinguishable from the point of view of p. That is why for set of states ρ and a player p, we need to consider the states that are observationally equivalent to p to states in ρ. The model checking algorithm in [9] is PTIME-complete. However, in our case for every player p, for every state q, we need to have $[q]_p$ pre-computed. This introduces $O(P.Q^2.\Pi)$ additional complexity for initial processing. If this is pre-computed for each player then computing necessary $[q]_A$s will be easy if efficient implementations are used. Also the $\mathcal{E}$ function adds $O(Q^2)$ complexity to the steps of the algorithm.

6.5.5 Semantics of ATL* with iNCGS

ATL formulae are somewhat restrictive in structure. One cannot express formulae like "a group of players P can guarantee that always if an accident happens, they will notify the police in the next step". This is due to the restriction that each temporal operator be preceded by a player coalition. However, one can express such a property using ATL*. Using ATL* the aforementioned property can be expressed as: $\langle\langle P \rangle\rangle \Box(\texttt{accidentHappened} \rightarrow \bigcirc(\texttt{policeInformed}))$. Given an iNCGS C, a sequence of states λ and an ATL* formula φ, we present the most interesting part of the semantics of ATL* using iNCGS:

Algorithm 1: ATLSymbolicModelChecking(C,φ)

```
Data: iNCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta \rangle$, ATL formula $\varphi$
Result: A set of states $S \subseteq Q$
begin
    foreach $\varphi' \in \mathrm{Sub}(\varphi)$ do
        case $\varphi' = p$
            $\Upsilon(\varphi') := \mathrm{Reg(p)}$
        case $\varphi' = \neg\varphi$
            $\Upsilon(\varphi') := Q \setminus \Upsilon(\varphi)$
        case $\varphi' = \varphi_1 \vee \varphi_2$
            $\Upsilon(\varphi') := \Upsilon(\varphi_1) \cup \Upsilon(\varphi_2)$
        case $\varphi' = \langle\!\langle A \rangle\!\rangle \bigcirc \psi$
            $\Upsilon(\varphi') := \mathcal{E}(\mathrm{Pre}(A, \mathcal{E}(\Upsilon(\psi), A)), A)$
        case $\varphi' = \langle\!\langle A \rangle\!\rangle \Box\psi$
            $\rho := Q$; $\tau = \Upsilon(\psi)$;
            while $\rho \not\subseteq \tau$ do
                $\rho = \rho \cap \tau$;
                $\tau := \mathcal{E}(\mathrm{Pre}(A, \mathcal{E}(\rho, A)), A) \cap \Upsilon(\psi)$
            $\Upsilon(\varphi') := \rho$
        case $\varphi' = \langle\!\langle A \rangle\!\rangle \psi_1 \mathcal{U} \psi_2$
            $\rho := \{\}$; $\tau = \Upsilon(\psi_2)$;
            while $\tau \not\subseteq \rho$ do
                $\rho = \rho \cup \tau$;
                $\tau := \mathcal{E}(\mathrm{Pre}(A, \mathcal{E}(\rho, A)), A) \cap \Upsilon(\psi_1)$
            $\Upsilon(\varphi') := \rho$
    return $\varphi$
```

- $C, q \models v \in \Pi$ if $v \in \pi(q)$.
- $C, q \models \neg\varphi$ if $C, q \not\models \varphi$
- $C, q \models \varphi_1 \vee \varphi_2$ if $C, q \models \varphi_1$ or $C, q \models \varphi_2$
- $C, q \models \langle\langle \mathcal{P} \rangle\rangle \psi$ if there exists a set of strategies $S_{\mathcal{P}}$ such that $\forall p \in \mathcal{P}. \forall q' \dot{\sim}_p q. \forall \lambda \in O(q', \mathcal{P}, S_{\mathcal{P}}). \forall \lambda' \ddot{\sim}_p \lambda. C, \lambda' \models \psi$
- $C, \lambda \models \neg\varphi$ if $\lambda \not\models \varphi$
- $C, \lambda \models \varphi_1 \vee \varphi_2$ if $C, \lambda \models \varphi_1$ or $C, \lambda \models \varphi_2$
- $C, \lambda \models \bigcirc\psi$ if $C, \lambda_1^\infty \models \psi$
- $C, \lambda \models \psi_1 \,\mathcal{U}\, \psi_2$ if $\exists j \geq 0. \forall 0 \leq i < j. C, \lambda_i^\infty \models \psi_1$ and $C, \lambda_j^\infty \models \psi_2$

6.6 Relating iNCGS to other Multi-agent Game Structures

In this section we show how other multi-agent game structures are related to iNCGS.

6.6.1 iTSGS and iNCGS

iNCGSs implement incomplete information by using equivalence classes of states. Each state belonging to an state equivalence class for a player a is indistinguishable to a. On the other hand $iTSGS$ uses a-views to identify the same concept of indistinguishable states. Following this observation, we define a view-to-equivalence class transformation function in order to relate the two different game structures. This function maps a-views in the iTSGS world to equivalence classes of states in the iNCGS world:

Definition The view-to-state set transformation function $\mathcal{V}$ transforms a given observable set of propositions v for a player p to the equivalence class of states where the viewable propositions hold: $\mathcal{V}(v, p) = [q]_p$ where $\pi_p(q) = v$.

Let $v = v_1, v_2, \ldots$ be sequence of views for a player p. Then the state sequences compatible with this view sequence for the player p is $\mathcal{T}(v, p) = \{\lambda \,|\, \forall i \geq 0.\ \lambda_i \in \mathcal{V}(v_i, p)\}$. This gives us a set of sequences where each state in the sequence belongs to the state equivalence class defined by the corresponding view. For any two traces λ_1 and λ_2 in $\mathcal{T}(v, p)$ we have that $\forall i \geq 0.\ \pi_p((\lambda_1)_i) = \pi_p((\lambda_2)_i) = v_i$.

We now define a function to translate a Turn Based Synchronous Game Structure ($S = \langle P, Q, \Pi, \pi, \sigma, R\rangle, \mathcal{P}$) into an iNCGS as follows:

Let $C : iTSGS \rightarrow iNCGS$ where $C((S = \langle P, Q, \Pi, \pi, \sigma, R\rangle, \mathcal{P})) = \langle P', Q', Q_0, \Sigma, \Pi', \pi', e, \dot{\sim}, \ddot{\sim}, \delta\rangle$ where we have:

- $P' = P$
- $Q' = Q$
- $Q_0 = Q$
- $\Pi' = \Pi$
- $\pi' = \pi$
- $\Sigma = \{\tau\} \cup 2^{2^Q}$
- $\forall q \in Q. \forall p \in P.$ if $\sigma(q) = p$ then $e_p(q) = \{[q']_p | \exists q'. R(q, q')\}$ else $e_p(q) = \{\tau\}$
- $\forall q \in Q. \forall \bar{v} \in \prod_{p \in P} e_p(q). \delta(q, \bar{v}) = \{q' | R(q, q') \wedge \forall p \in P. \bar{v}|p \neq \{\tau\} \rightarrow q' \in \bar{v}|_p(q)\}$
- For each player $p \in P$, $\forall q_1, q_2 \in Q.$ if $\pi(q_1) \cap \mathcal{P}_p = \pi(q_2) \cap \mathcal{P}_p$ then $q_1 \dot{\sim}_p q_2$.
- For each player $p \in P$, $\forall \lambda_1, \lambda_2 \in Q^+.$ if $|\lambda_1| = |\lambda_2|$ and $\forall 0 \leq i \leq |\lambda_1|. \lambda_{1,i} \dot{\sim}_p \lambda_{2,i}$ then $\lambda_1 \ddot{\sim}_p \lambda_2$.

We now define a function $\mathcal{F} : (V_a^+ \rightarrow V_a) \rightarrow (Q^+ \rightarrow \Sigma)$ that will transform an iTSGS strategy into an iNCGS strategy.

Definition The function $\mathcal{F}(f_a) = S_a$ mapping an iTSGS strategy for a player a, $f_a : V_a^+ \rightarrow V_a$ to S_a is an iNCGS strategy for the player a where S_a is defined as follows: for each view sequence $v = v_0, v_1, \ldots, v_n$, for each trace sequence $\lambda \in \mathcal{T}(v, a)$, if $\sigma(\lambda_{|\lambda|}) \neq a$ then $S_a(\lambda) = \tau$. On the other hand if $\sigma(\lambda_{|\lambda|}) = a$, then $S_a(\lambda) = [q]_a$ where $\mathcal{V}(f_a(v), a) = [q]_a$.

Notice that the iNCGS strategy S_a suggests equivalence classes of states which is the action set Σ for the iNCGS obtained by translating the iTSGS. Given a state $q \in Q$, a set A of players and a set F_A of strategies, one for each player in A, a computation $\lambda = q_0, q_1, q_2, \ldots$ is an outcome in *out*(q, F_A) if $q = q_0$ and for all $i \geq 0$, if $\sigma(q_i) = a \in A$ then $\pi_a(q_{i+1}) = f_a(\pi_a(\lambda[0, i]))$. This states that a trace will be in the outcome of following a strategy if at every point along the trace, if it is the player's turn then the next state has the same view as suggested by the strategy. In the case of the iNCGS strategies, let us consider a collection of strategies S_A for a group of players A. A computation $\lambda = q_0, q_1, q_2, \ldots$ is an outcome in *out*(q, S_A) if $q = q_0$ and for all $i \geq 0$, $q_{i+1} \in \delta(q_i, \bar{v})$ for some $\bar{v} \in \prod_{p \in P} e_p(q_i)$ with $\bar{v}|_A = S_A(\lambda_{0,i})$. Now, we present a theorem correlating the outcomes of iTSGS and iNCGS strategies.

Theorem 6.6.1 *Let $T = (S = \langle P, Q, \Pi, \pi, \sigma, R \rangle, \mathcal{P})$ be a Turn Based Synchronous Game Structure. Let $C((S, \mathcal{P})) = I$ where $I = \langle P', Q', Q_0, \Sigma, \Pi', \pi', e, \dot{\sim}, \ddot{\sim}, \delta \rangle$. Let, $A \subseteq P$ be a group of players. Let F_A be a collection of iTSGS strategies and $S_A = \mathcal{F}(F_A)$ be the translated strategy. Let $q \in Q$ be a state. Then a computation $\lambda \in O_T(q, F_A) \implies \lambda \in O_I(q, F_A)$. Here $O_T(q, F_A)$ and $O_I(q, S_A)$ are the sets of computations of T and I obtained by following F_A and S_A respectively.*

Proof Let us consider an arbitrary trace $\lambda \in Q^+$ where $\lambda \in O_T(q, F_A)$. The proof is by induction on the length of λ. If $|\lambda| = 0$, then $\lambda = q$. Since the computations are rooted at q, we will have that $\lambda \in O_I(q, S_A)$. Now, let us consider an arbitrary position i in λ up to which the theorem holds.

Let us first consider the case where $\sigma(q_i) \notin A$. Then by the definition of C, $\forall a \in A. e_a(q_i) = \tau$. And the destination state is not influenced by the decisions of the players in A. Then since $\lambda \in O_T(q, F_A)$, we have that $R(q_i, q_{i+1})$. Let there exists a player $b \in P$ and $b \notin A$, such that $\sigma(q_i) = b$. Since the strategy F_A does not influence the action options for b, b will be able to choose $[q_{i+1}]_b$ while the players in A are acting according to S_A. Then since $q_{i+1} \in [q_{i+1}]_b$, we will have $q_{i+1} \in \delta(q_i, \bar{v})$ where $\bar{v}|_b = [q_{i+1}]_b$ and $\bar{v}|_A = S_A(\lambda_{0,i})$. Thus if $\lambda.q_{i+1} \in O_T(q, F_A)$ then $\lambda.q_{i+1}$ will also be in $O_I(q, S_A)$.

Now, let us consider the more interesting case of $\sigma(q_i) = a \in A$. Now, according to the definition of C, we have that for all states to which there was a possible transition from q_i in T, the equivalence classes of a for those states are included in $e_a(q_i)$. Moreover, for all possible action vectors $\bar{v}$ in q_i, we will have that $\bar{v}|_p = \tau$ for all players $p \in P \setminus \{a\}$. And by the definition of C, we have that each state belonging to the equivalent classes $[q]_a$ in $e_a(q_i)$ is a possible outcome in q_i if that particular $[q]_a$ is chosen by a. Now by the definition of $\mathcal{F}$, we have that if $f_a(q, \lambda_{0,i}) = q_{i+1}$ then we have that $S_a(\lambda_{0,i}) = [q_{i+1}]_a$. Now since $q_{i+1} \in [q_{i+1}]_a$, from the definition of C we have that $q_{i+1} \in \delta(q, \bar{v})$ where $\bar{v}|_a = [q_{i+1}]_a$. Hence we will have that $\lambda.q_{i+1} \in O(q, S_A)$.

We now present ATL semantics using iTSGS to show that our translation preserves it too.

ATL Semantics using iTSGS

Given a state $q \in Q$, a set $A \subseteq P$ of players, and a set $F_A = \{f_a | a \in A\}$ of strategies one for each player in A, a computation $\lambda = q_0, q_1, q_2, \ldots$ is in an outcome in $out(q, F_A)$ if $Q_0 = q$ and for all positions $i \geq 0$, if $\sigma(q_i) \in A$, then $\pi_a(q_{i+1}) = f_a(\pi_a(\lambda[0, i]))$ for $a = \sigma(q_i)$. We now present the semantics of an ATL formula φ using an iTSGS T. Let q be a state in T.

- $\varphi = p \in \Pi$. Then $q \models_T \varphi$ iff $p \in \Pi(q)$.
- $q \models_T \varphi_1 \vee \varphi_2$ iff $q \models_T \varphi_1$ or $q \models_T \varphi_2$
- $q \models_T \neg\varphi$ iff $q \not\models_T \varphi$
- $q \models_T \langle\langle A \rangle\rangle \bigcirc \varphi$ iff either $\sigma(q) \in A$ and there exists a $\sigma(q)$-view $v \in \Pi_{\sigma(q)}$ such that for all states q' with $R(q, q')$ and $\pi_{\sigma(q)}(q') = v$ and $q' \models_T \varphi$ or $\sigma(q) \notin A$ and for all states q' with $R(q, q')$ we have that $q' \models_T \varphi$.
- $q \models_T \langle\langle A \rangle\rangle \Box\varphi$ iff either $\sigma(q) \in A$ and there exists a $\sigma(q)$-view $v \in \Pi_{\sigma(q)}$ such that for all states q' with $R(q, q')$ and $\pi_{\sigma(q)}(q') = v$ and $q' \models_T \langle\langle A \rangle\rangle \Box\varphi$; $\sigma(q) \notin A$ and for all states q' with $R(q, q')$ we have that $q' \models_T \langle\langle A \rangle\rangle \Box\varphi$.

- $q \models_T \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a set of strategies F_A such in all traces $\lambda \in O(q, F_A)$, we have that there exists a j such that $\lambda_j \models_T \varphi_2$ and for all $0 \leq i < j$, $\lambda_i \models \varphi_1$.

6.6.2 ATL Interpretation using iTSGS carries over to iNCGS

In [9], only well formed ATL formulae are considered. Given a turn based synchronous game structure with incomplete information a formula of the form $\langle\!\langle A \rangle\!\rangle \psi$ is well formed if the propositions mentioned in ψ are a subset of the propositions that can be observed by players in A. Now an iTSGS models an ATL formula $\langle\!\langle A \rangle\!\rangle \phi$ if there exists a strategy for the players in A such that in all ensuing views ϕ holds. We can translate it that an iNCGS will model $\langle\!\langle A \rangle\!\rangle \phi$ if there exists a strategy for the players in A such that in all ensuing outcomes ϕ holds. We define the function $\mathcal{E} : Q^+ \to (2^Q)^+$ where $\mathcal{E}(q_0, q_1, q_2, \ldots, p) = [q_0]_p, [q_1]_p, [q_2]_p, \ldots$. The outcome of following a strategy S_a by a player p at a state q is defined as the set of traces $\{\lambda | \lambda_0 = q \wedge \forall i \geq 0.\ \exists \bar{v}.\ \lambda_{i+1} \in \delta(q, \bar{v}) \wedge \lambda_{i+1} \in S_p(\mathcal{E}(\lambda, p))\}$. By our translation when a trace $\lambda = q_0, q_1, \ldots \in O_T(q, F_A)$ then $\lambda = q_0, q_1, \ldots \in O_I(q, F_A^I)$ where $F_A^I = \mathcal{F}(F_A)$.

Theorem 6.6.2 *Let φ be an ATL formula. Let $T = \langle S, \mathcal{P} \rangle$ be an iTSGS and $I = C(T)$ be an iNCGS. Let q be state in T. Then $q \models_T \varphi$ iff $q \models_I \varphi$.*

The inductive hypothesis is that $\forall q \in Q.\ q \models_T \varphi \to q \models_I \varphi$. The base case is when φ is an atomic proposition. Now we discuss the steps of structural induction where ATL formulae is built from φ using the structural operators:

Proof The proof is by induction on the structure of φ.

- $\varphi = v \in \Pi$. C maps the states in T to states in I using the identity function. It does the same for π. Hence if we have a proposition $v' \in \pi_T(q)$ then it is also the case that $v' \in \pi_I(q)$. Thus we will have that if a state $q \in Q \models_T p$ then $q \in Q \models_I p$.

- Let φ_1 and φ_2 be two ATL formulae such that $q \models_T \varphi_1$ or $q \models_I \varphi_1$, $q \models_T \varphi_2$, $q \models_I \varphi_2$. Then we have that $q \models_I \varphi_1 \vee \varphi_2$ by the induction hypothesis.

- $\langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ There exists a set of strategies F_A such that all state sequences $\lambda \in O(q, F_A)$ we have that $\lambda_1 \models \varphi$. $q \models_T \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ iff either $\sigma(q) \in A$ and there exists a $\sigma(q)$-view $v \in \Pi_{\sigma(q)}$ such that for all states q' with $R(q, q')$ and $\pi_{\sigma(q)}(q') = v$ we have $q' \models_T \varphi$ or $\sigma(q) \notin A$ and for all states q' with $R(q, q')$ we have that $q' \models_T \varphi$. In this case from the definition of the translation, there will be transitions of the form $q' \in \delta(q, \bar{v})$ where $\bar{v}|_{\sigma(q)} = [q']_p$. Also due to the translation of strategies, the translated strategy will suggest $[q']_p$ resulting in states in $[q']_p$ to be chosen by the transition function. And since in each of the states in $[q']_p$ the property holds by the induction hypothesis it will hold in the translated iNCGS also. If $\sigma(q) = p \in A$ then $e_p(q) = \{[q']_p | R(q, q')\}$. Then we have that $S_a \in [q']_p$ for some q'. Combining with the definition of δ in the translation and the definition of outcomes, we get for all traces

$\lambda \in out(q, S_a)$ will have that $\lambda_1 \in [q']_p$. Now for all traces $\lambda \in out(q, S_a)$ we will have that $\lambda_1 \models_I \varphi$ using the induction hypothesis. Similarly when $a \notin \sigma(q)$, we have $\lambda_1 \models_I \varphi$ using the induction hypothesis. Now, in the iTSGS world, if $q \models_I \langle\langle A \rangle\rangle \bigcirc \varphi$, then there exists a strategy S_A such that for all computations $\lambda \in O(q, S_A)$, we have $\lambda_1 \models \varphi$. Then let us consider the case where we have that there exists a player $p \in A$ such that $\sigma(q) = p$ and $[\lambda_1]_p \in e_P(\lambda_0)$. Then we have that $R(\lambda_0, \lambda_1)$. Also, we can have a strategy F_p for the player p such that $\pi_p(\lambda_1) = f_p(q, F_p)$ which will help satisfy $\langle\langle A \rangle\rangle \bigcirc \varphi$. The case where the turn does not belong to the players in A is very similar.

6.6.3 iCGS

Schobbens presents iCGS (incomplete information CGS) in [103]. They also use state equivalence classes to model incomplete information. There are some differences with iNCGS. The transition system of iNCGS is nondeterministic as opposed to deterministic transition system of iCGS. We have notions of trace equivalence classes in addition to state equivalence classes. They consider imperfect information about traces while considering strategies. Their motivation for opting for this approach is that only the strategies need to consider trace equivalences. However, we wanted a more generalized notion of trace equivalence than theirs. They also provide a classification of strategies based on the level of incognizance about trace differences. Let us show how we can encode the four different types of strategies as presented in [103] into iNCGS strategies with deterministic δ or *iDCGS*. A strategy for a player p is a function $f_p : T \to \Sigma$ where different types of strategies can be obtained in the iCGS world for different type of T.

- When $T = Q^+$, then they refer to the strategy as perfect *I*nformation and perfect *R*ecall strategy. This type of strategies can distinguish between every step of two traces of equal length and has no information hiding whatsoever. However, what if $\sim_p$ is not an identity? Should p's strategy still be able to distinguish among all states? This issue is not clear from their work. Assuming that they mean for the player's to have complete information, we translate iCGS$_{IR}$ strategies to *iDCGS* strategies as follows: if we have for all players p, $\dot{\sim}_p$ and $\ddot{\sim}_p$ are equalities then *iCGS* strategies becomes *iDCGS* strategies.

- *iCGS* strategies with perfect *I*nformation and imperfect *r*ecall have $T = Q$ that is they can only consider the last step of a trace and for that last step state they have perfect information. If for all players p, we have $\dot{\sim}_p$ be equality and $\forall \lambda, \lambda' \in Q^+ . \lambda \ddot{\sim}_p \lambda'$ if $\lambda_0 = \lambda'_0$ then it is the same as iCGS strategy with perfect information and imperfect recall.

- *iCGS* strategies with imperfect *i*nformation and perfect *R*ecall have $T = [Q]_p^+$. If for all players p, we have $\forall \lambda, \lambda' \in Q^+ . \lambda \ddot{\sim}_p \lambda'$ if $|\lambda| = |\lambda'|$ then it is the same as iCGS strategy with imperfect information and perfect recall.

- *iCGS* strategies with perfect *i*nformation and imperfect *r*ecall have $T = [Q]_p$ that is they can only consider the last step of a trace and for that last step state they have imperfect view. If for all players p, $\forall \lambda, \lambda' \in Q^+. \lambda \ddot{\sim}_p \lambda'$ if $[\lambda_0]_p = [\lambda'_0]_p$ then it is the same as iCGS strategy with imperfect information and imperfect recall.

This illustrates the use of having $\ddot{\sim}$ in addition to $\dot{\sim}$. One can define $\ddot{\sim}$ in many different ways and achieve the interpretation they want. $\ddot{\sim}$ may even not require that state sequences be of equal length. Then we can have stuttering invariance of sequences. We can make different sequences be equivalent if they have visited a certain set of states in their journey.

Our semantics for ATL* varies from their semantics for ATL* in the handling of player modality. Let us consider their modeling of ATL* semantics where the ATL* formula is: $\langle\langle A \rangle\rangle \varphi$. Then the ATL* semantics provided in [103] states that given a sequence of states λ

- $\lambda \models \langle\langle A \rangle\rangle \varphi$ iff there exists a set of XY strategies F_A, one for each player $p \in A$, such that $\forall a \in A. \forall q' \dot{\sim}_a \lambda_0. \forall \lambda' \in O(q', F_A)$ we have $\lambda' \models \varphi$.

iNCGS semantics for the same formula is:

- $\lambda \models \langle\langle A \rangle\rangle \varphi$ iff there exists a set of XY strategies F_A, one for each player $p \in A$, such that $\forall a \in A. \forall q' \dot{\sim}_{\mathbf{A}} \lambda_0. \forall \lambda' \in O(q', A, F_A)$ we have $\lambda' \models \varphi$.

The difference in semantics is that given a formula $\langle\langle A \rangle\rangle \varphi$, we consider a more constrained view of the states from which the ensuing traces need to satisfy φ. According to our definitions, $q \dot{\sim}_A q'$ if $\forall a \in A.\ q \dot{\sim}_a q'$. Our contention is that, given a set of agents, two states are indistinguishable from the point of view of that set of agents if no agent in that group can distinguish among them. Let us consider the CGS presented in Figure 6.1 and the equivalence classes presented in Section 6.5.2. Let us assume that `AirCraftAPilot` cannot distinguish among the states q_1 and q_2 because presumably he cannot observe `AirCraftBDetCollision` proposition. Similarly, `AircraftBPilot` cannot distinguish among states q_2 and q_3. Then the equivalence classes for `AirCraftAPilot` can look like: $\{\{q\}, \{q_1, q_2\}, \{q_3\}, \ldots\}$ and the equivalence classes for `AircraftBPilot` can look like: $\{\{q\}, \{q_1\}, \{q_2, q_3\}, \ldots\}$. Now if we are trying to evaluate $q_2, q_3, q_4 \models \langle\langle$ `AirCraftAPilot`, `AircraftBPilot` $\rangle\rangle \varphi$, then for a pair of strategies for the two pilots, iCGS semantics will consider traces emanating from q_1 because, $q_1 \dot{\sim}_{\texttt{AirCraftAPilot}} q_2$, traces emanating from q_3 because, $q_2 \dot{\sim}_{\texttt{AircraftBPilot}} q_3$ and of course the traces emanating from q_2. Our semantics will only consider the traces emanating from state q_2.

6.6.4 Game Arenas

Dima *et al* present model checking ATL with knowledge, imperfect information, perfect recall and communicating coalitions in [43]. They interpret ATL formulae over *game arenas*. A game arena $\Gamma = \langle Ag, Q, (\Sigma_a)_{a \in Ag}, \delta, Q_0, (\Upsilon_a)_{a \in Ag}, \gamma \rangle$ is defined as:

- Ag is a set of agents
- Q is a set of states
- Σ_a is a finite set of actions for agent a. We use Σ_A to denote all tuples of action choices for the players in A: $\Sigma_A = \prod_{a \in A}(\Sigma_a)$. We will also use $\Sigma = \Sigma_{Ag}$.
- $Q_0 \subseteq Q$ is the set of initial states
- Υ_a is a finite set of propositions observable by agent a. $\Upsilon_A = \bigcup_{a \in A} \Upsilon_a$ and $\Upsilon = \Upsilon_{Ag}$
- $\gamma : Q \to 2^{\Upsilon}$ is the state labeling function
- $\delta : Q \times \Sigma \to (2^Q \setminus \{\})$

They use $q \overset{c}{\to} r$ for transitions $(q, c, r) \in \delta$. Now, we show how to convert an iNCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta_{iNCGS} \rangle$ into a game arena $\Gamma = \langle Ag, Q, (\Sigma_a)_{a \in Ag}, \delta, Q_0, (\Upsilon_a)_{a \in Ag}, \gamma \rangle$.

Definition Let $C = \langle P, Q, Q_0, \Sigma_{iNCGS}, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta_{iNCGS} \rangle$ be an iNCGS. Then we define a conversion function $\mathcal{G} : iNCGS \to$ Game Arena where $\mathcal{G}(C) = \langle Ag, Q, (\Sigma_a)_{a \in Ag}, \delta, Q_0, (\Upsilon_a)_{a \in Ag}, \gamma \rangle$ is defined as follows:

- $Ag \ = \ P$
- $Q \ = \ Q \cup \{q_s\}$
- for each agent $a \in Ag$, $\Sigma_a = \bigcup_{q \in Q} e_a(q)$
- $Q_0 = Q$
- $\forall a \in Ag. \Upsilon_a = \{\zeta \in \Pi \,|\, \forall q_1, q_2 \in Q. q_1 \dot{\sim}_a q_2 \to (\zeta \in \pi(q_1) \leftrightarrow \zeta \in \pi(q_2)\}$
- $\forall q \in Q. \gamma(q) = \pi(q)$ and $\pi(q_s) = \{\}$
- for each state $q \in Q$, for each $\bar{c} \in \Sigma_{Ag}$, if $\forall a \in Ag. \bar{c}|_a \in e_a(q)$ then $\forall q' \in \delta_{iNCGS}(q, \bar{c}). (q, \bar{c}, q') \in \delta$ else $(q, \bar{c}, q_s) \in \delta$.

In this translation q_s is a sink state. Given an arena Γ, a run ρ is a sequence of transitions $q'_i \xrightarrow{c_i} q_i''$ where for all i, $q'_{i+1} = q_i''$. Given a run $\rho = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \ldots$, q_i is denoted by $\rho[i]$ and c_{i+1} is denoted by $act(\rho, i)$. Runs ρ and ρ' are indistinguishable from the point of view of a coalition of agents $A \subseteq Ag$ denoted $\rho \sim_A \rho'$ if $|\rho| = |\rho'|$, $act(\rho, i)|_A = act(\rho', i)|_A$ for all $i < |\rho|$, and $\gamma_A(\rho[i]) = \gamma_A(\rho'[i])$ for all $i \leq |\rho|$. Here $\gamma_A(q) = \gamma(q) \cap \Upsilon_A$. We denote $\gamma_A(\rho) = \gamma_A(\rho[0]), \gamma_A(\rho[1]), \ldots$.

A strategy for a coalition A in a game arena is a function $\sigma : (2^{\Upsilon_A})^* \to \Sigma_A$. A strategy σ_A is *compatible* with a run ρ if for all $i \leq |\rho|$, we have $\sigma_A(\gamma_A(\rho[0, i])) = c_{i+1}|_A$. Thus game arena agent strategies are deterministic. A deterministic strategy F_A for a group of players $A \in P$ in an iNCGS is a collection of strategies where each strategy is a function $F_a : Q^+ \to \Sigma$. A set of strategies F_A can be converted to a strategy for a coalition A, where for every sequence of states λ, $\sigma_A(\gamma_A(\lambda)) = \prod_{a \in A} F_a(\lambda)$. Every computation $\lambda \in O(q, A, F_A)$ will correspond to a run $\rho = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \ldots = \mathcal{R}(\lambda, A, F_A)$ where for all $i \geq 0$, we have $q_0 \in Q_0$, $q_i \in [\lambda_i]_A$, $c_i|_A = F_A(\lambda_i)$. ρ will be compatible with σ_A by definition of compatibility.

[43] considers the logic ATL^D_{iR}: distributed knowledge, incomplete information and perfect recall ATL. The syntax of is presented below:

$$\varphi = p \in \Pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\!\langle A \rangle\!\rangle \bigcirc \xi \mid \langle\!\langle A \rangle\!\rangle \xi_1 \,\mathcal{U}\, \xi_2 \mid \langle\!\langle A \rangle\!\rangle \xi_1 \,\mathcal{W}\, \xi_2 \mid K_A(\varphi)$$

ATL syntax is contained in ATL^D_{iR} syntax. Satisfaction of an ATL formula with respect to a given game arena Γ, an infinite run ρ with $\rho[0] \in Q_0$ and a position i in ρ can be excerpted from the ATL^D_{iR} semantics provided in [43] as:

- $(\Gamma, \rho, i) \models p$ if $p \in \gamma(\rho[i])$.
- $(\Gamma, \rho, i) \models \varphi_1 \wedge \varphi_2$ if $(\Gamma, \rho, i) \models \varphi_1$ and $(\Gamma, \rho, i) \models \varphi_2$.
- $(\Gamma, \rho, i) \models \neg\varphi$ if $(\Gamma, \rho, i) \not\models \varphi$.
- $(\Gamma, \rho, i) \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ if there exists a strategy σ_A such that $(\Gamma, \rho', i+1) \models \varphi$ for all infinite runs ρ' such that $\rho'[0] \in Q_0$ which are compatible with σ_A and satisfy $\rho'[0..i] \sim_A \rho[0..i]$.
- $(\Gamma, \rho, i) \models \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a strategy σ_A such that for all infinite runs ρ' such that $\rho'[0] \in Q_0$ which are compatible with σ_A and satisfy $\rho'[0..i] \sim_A \rho[0..i]$, there exists $j \geq i$ such that $(\Gamma, \rho', j) \models \varphi_2$ and $(\Gamma, \rho', k) \models \varphi_1$ for all $k = i, \ldots, j-1$.

With all the necessary concepts in place we now show that if an ATL formula φ holds of a state $q \in Q$ in the iNCGS then it will hold of traces starting at q in the translated game arena.

Theorem 6.6.3 *Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta_{iNCGS} \rangle$ be an iNCGS. Let $\mathcal{G}(C) = \Gamma$ where $\Gamma = \langle Ag, Q, (\Sigma_a)_{a \in Ag}, \delta, Q_0, (\Upsilon_a)_{a \in Ag}, \gamma \rangle$. Let φ be an ATL formula. Then $q \models \varphi$ if $(\Gamma, \rho, 0) \models \varphi$ where ρ is any infinite run of Γ with $\rho[0] = q$.*

Proof The proof is by induction on the structure of φ.

- $\varphi = \upsilon \in \Pi$. Then if $q \models \upsilon$, then $\upsilon \in \pi(q)$ and hence $\upsilon \in \gamma(\rho[0])$. Then $(\Gamma, \rho, 0) \models \upsilon$ as $\upsilon \in \gamma(\rho[0])$.
- Similar reasoning works for $\varphi = \varphi_1 \wedge \varphi_2$ and $\varphi = \neg\varphi'$.
- $q \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ if there exists a set of strategies F_A such that for all computations $\lambda \in O(q, A, F_A)$, $\lambda_1 \models \varphi$. Since every set of strategies F_A can be translated into an equivalent

strategy σ_A such that every $(\Gamma, \rho', i+1) \models \varphi$ for all infinite runs ρ' with $\rho'[0] \in Q_0$ which are compatible with σ_A and satisfy $\rho'[0..i] \sim_A \rho[0..i]$. Then we have that there exists a strategy σ_A such that every infinite initialized run ρ' compatible with σ_A, we have $(\Gamma, \rho', 1) \models \varphi$ due to inductive hypothesis.

- $q \models \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2$, $q \models \langle\!\langle A \rangle\!\rangle \Box \varphi$ proof is similar.

This proof allows us to provide an alternate proof that there exists an algorithm for determining whether a state in an iNCGS C satisfies an ATL formula φ. We can use the semantics preserving reduction of iNCGS into a game arena and conclude that since there exists an algorithm for model checking ATL formula satisfaction with respect to a game arena, there exists an algorithm for checking ATL formula satisfaction with respect to an iNCGS.

6.7 Variations of iNCGS

iNCGS has the capability to encode both non-deterministic transition system and incomplete observational capability of players. We now show how we can obtain different variations of an iNCGS. Let us assume we have an iNCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta \rangle$.

NCGS A *N*ondeterministic Concurrent Game Structure(NCGS) with complete observational capability of the players can be obtained by restricting the state equivalence classes and the trace equivalence classes to be identities. For all $p \in P$, for all $q_1, q_2 \in Q$, we must have that $q_1 \dot{\sim}_p q_2$ iff $q_1 = q_2$. Similarly, for all $p \in P$, for all $\lambda_1, \lambda_2 \in Q$, we must have that $\lambda_1 \ddot{\sim}_p \lambda_2$ iff $\lambda_1 = \lambda_2$. Two state sequences λ_1 and λ_2 are the same if $|\lambda_1| = |\lambda_2|$ and $\forall i \geq 0.\ \lambda_{1(i)} = \lambda_{2(i)}$

DCGS A *D*eterministic Concurrent Game Structure(DCGS) with complete observational capability of the players can be obtained by restricting the state equivalence classes and the trace equivalence classes to be identities and adding the constraint to make all the transitions deterministic: $\forall q \in Q.\ \forall \bar{v} \in \prod_{p \in P} e_p(q).\ |\delta(q, \bar{v})| = 1$. If Q_0 is a singleton then DCGSs in fact coincide with CGSs as presented in [9].

iDCGS A *i*ncomplete information *D*eterministic Concurrent Game Structure(DCGS) with *incomplete* observational capability of the players can be obtained by just adding the constraint to make all the transitions deterministic: $\forall q \in Q.\ \forall \bar{v} \in \prod_{p \in P} e_p(q).\ |\delta(q, \bar{v})| = 1$.

6.8 Strategic Modeling of Human Behavior

We focus on obtaining a suitable formalism for modeling human operator behavior. On this track we now present another method of modeling human behavior. We present a semantics of ATL where a group of human operators has decided on a course of action and wants to know whether their plan of action can help them maintain a desired property. We find that modeling human operator behavior through the e function of a CGS or an iNCGS can be too simplistic because the e function is memoryless. Given a current state of affairs, using e function as behavior model, the human operators only rely on the current state to offer a set of possible actions. However, in real life, humans may vary their behavior based on their previous experience, guidance and training. Let us consider the supermarket checkout scenario presented in Section 4.4. Here in the initial state, the customer needs to press the `Start` button on the touch-screen display. Then they are supposed to scan each item and then put it in the bags in the bagging area. The system can detect if an item has been placed in the bagging area before having been scanned. Thus at this state, bagging an item is a safe but not recommended action. Now let us assume the customer has never used the system before and is a little confused about the order in which he should perform the necessary actions. He picks up the first item and instead of placing it in front of the scanner, he places it inside a bag in the bagging area. The scale detects the weight of an unscanned item and alerts the attendant. The attendant comes in and puts the item back in the cart and restores the system to the previous state. Now this time, the customer is aware of the correct sequence of steps he needs to perform and from now on at this state he only chooses to perform the scanning of an item action. Thus his behavior changed with his experience. he enabled action function e of a CGS will always offer the same set of actions to the customer. He can always choose between scanning or bagging an item. One can attempt to remedy this situation with having two copies of the state, one where a proposition indicating that the customer has had such an experience is set to be true and one where such a proposition is set to be false. However, we would like to have an easier and more succinct method of modeling history dependent human behavior.

6.8.1 Human Behaviors as Strategies

A strategy for a player is a game plan that prescribes actions to him at each state depending upon the states that have already been traversed in order to reach that state. The type of strategies is $Q^+ \to \Sigma$. Thus given a history of states, the strategy suggests the next move to a player. The strategies are a better model for human behaviors as they provide a mechanism to model history dependent human action choices. Let us consider the supermarket self-scan checkout scenario situation again. Let us consider a CGS corresponding to the scenario as presented (partial) in Figure 6.2. In state q_0, the customer presses the `Start` button and the CGS moves to state q_1. Now, the customer may bag an item or scan an item. If he puts the item in the bag then an alarm is raised by the scale. The attendant can remedy the situation and make the self-scan system return to state q_1. But from now on, the customer does not want any repeat of the alarm incident and always scans an item first. Thus the behavior of this customer is that given a history of just q_0, q_1, he may choose between bagging an item and scanning an item. But given a history of q_0, q_1, q_2, q_1, he will only choose the action of scanning an item. Thus we can describe this human behavior with a strategy $S_{\texttt{customer}}$

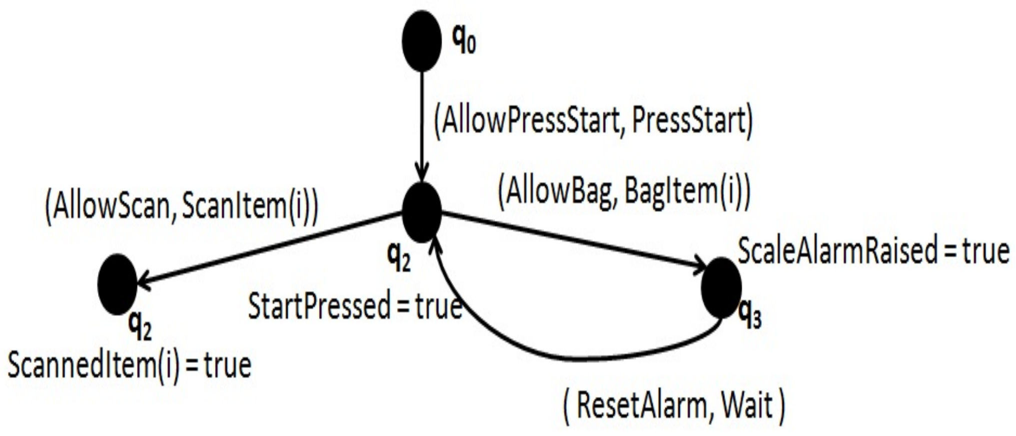


Figure 6.2: CGS for self-scan checkout scenario

where $S_{\texttt{customer}}(q_0) = \texttt{PressStart}$, $S_{\texttt{customer}}(q_0, q_1) = \texttt{BagItem(i)}$, $S_{\texttt{customer}}(q_0, q_1, q_2, q_1) = \texttt{ScanItem(i)}$.

We are also interested in determining what behavior guarantees human behaviors should provide in order to ensure safe operation for the computer system. Given a human behavior we may ask the whether the human behavior conforms to the protection envelope guarantee. One may ask: given a human behavior, can it help maintain system safety? This gives rise to the concept of a human behavior strategy satisfying an ATL property in an iNCGS. Moreover, we notice that in the original semantics of ATL, once a strategy is chosen for a player it is not fixed for the player. A player may change his strategy whenever a coalition involving him is encountered in a formula. Thus we cannot reason about a single strategy modeling a behavior using the usual ATL semantics. We need to perform irrevocable strategy modeling of behaviors in the manner of Ågotnes *et al* in [7]. In order to help ascertain whether a behavior conforms to a guarantee, we present an alternate semantics of ATL formulae.

6.8.2 Strategic Semantics of ATL using iNCGS

We call the semantics of ATL formulae defined over player strategies in iNCGS: *strategic semantics*. This semantics deviates from the usual concept of ATL semantics using iNCGS in these ways: (i) it provides an understanding of whether *a strategy* for a set of players are sufficient for satisfying an ATL formula, (ii) it affixes strategies to players: once a strategy is chosen for a player the strategy can not be revoked or modified during the rest of the analysis. Essentially, this provides us with a mechanism to assess the "goodness" of a strategy. A strategy for a group of players is good if the group of players can help maintain a guarantee by following the strategy. Given an iNCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \dot{\sim}, \ddot{\sim}, \delta \rangle$, a state $q \in Q$, a set of players $\mathcal{P} \subseteq P$, a set of strategies $S_{\mathcal{P}}$

containing one strategy for each player in $\mathcal{P}$, a state q and an ATL formula φ, we now present strategic semantics:

- $\mathcal{P}, C, q, S_{\mathcal{P}} \models v \in \Pi$ if $v \in \pi(q)$.
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models \neg\varphi$ if $\mathcal{P}, C, q, S_{\mathcal{P}} \not\models \varphi$
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models \varphi_1 \vee \varphi_2$ if $\mathcal{P}, C, q, S_{\mathcal{P}} \models \varphi_1$ or $\mathcal{P}, C, q, S_{\mathcal{P}} \models \varphi_2$
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models \langle\!\langle \mathcal{P}' \rangle\!\rangle \bigcirc \psi$ if there exists a set of strategies $S_{\mathcal{P}'\setminus\mathcal{P}}$ such that $\forall q \in [q]_a$. $\forall \lambda \in O(q', C, S_{\mathcal{P}} \cup S_{\mathcal{P}'})$. $\forall \lambda' \ddot{\sim}_{\mathcal{P}} \lambda$. $\mathcal{P} \cup \mathcal{P}', C, \lambda'_1, S_{\mathcal{P}} \cup S_{\mathcal{P}'} \models \psi$
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models \langle\!\langle \mathcal{P}' \rangle\!\rangle \Box\psi$ if there exists a set of strategies $S_{\mathcal{P}'\setminus\mathcal{P}}$ such that $\forall q' \in [q]_a$. $\forall \lambda \in O(q', C, S_{\mathcal{P}} \cup S_{\mathcal{P}'})$. $\forall \lambda' \ddot{\sim}_{\mathcal{P}} \lambda$. $\forall i \geq 0$. $\mathcal{P} \cup \mathcal{P}', C, \lambda'_i, S_{\mathcal{P}} \cup S_{\mathcal{P}'} \models \psi$
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models \langle\!\langle \mathcal{P}' \rangle\!\rangle \psi_1 \,\mathcal{U}\, \psi_2$ if there exists a set of strategies $S_{\mathcal{P}'\setminus\mathcal{P}}$ such that $\forall q' \in [q]_a$. $\forall \lambda \in O(q', C, S_{\mathcal{P}} \cup S_{\mathcal{P}'})$. $\forall \lambda' \ddot{\sim}_{\mathcal{P}} \lambda$. $\exists j \geq 0$. $\forall 0 \leq i < j$. $\mathcal{P} \cup \mathcal{P}', C, \lambda'_i, S_{\mathcal{P}} \cup S_{\mathcal{P}'} \models \psi_1$ and $\mathcal{P} \cup \mathcal{P}', C, \lambda'_j, S_{\mathcal{P}} \cup S_{\mathcal{P}'} \models \psi_2$

Now we can ask of a human behavior or any strategy of any player in an iNCGS being sufficient for satisfying an ATL property. We have not yet explored the feasibility of model checking strategic ATL semantics in iNCGS. However, we believe model checking for the strategic semantics for ATL can be achievable following the work of Ågotnes *et al.* They presented ATL with irrevocable strategies, IATL and MIATL in [7]. There they perform tree unfolding of CGSs and update the unfolded CGS according to chosen irrevocable strategies. However, even in the CGS setting, model checking ATL with irrevocable strategies are NP-complete.

6.9 Alternative Semantics for ATL*

We have presented an alternate method for modeling human behavior and judging whether that behavior will be sufficient in maintaining protection envelope guarantees. Let us first focus on human behavior. Behavior is the choice of action in a scenario. The same human may choose different actions in the same scenario depending upon how he arrived at that scenario. This led us to considering strategies as models for human behavior. However, the strategies considered this far are prescriptive in nature. For every history and a current state, the strategy suggests only one next possible move. However, human behavior may not be so deterministic in nature. Let us consider the example scenario presented in Section 6.2. `AirCraftAPilot` had information from both the `ATC` and his own aircraft. The `ATC` recommendation was to descend and `AircraftA` recommendation was to ascend. However, he was unsure of the capability of his aircraft being able to operate safely in higher altitudes and decided to follow the `ATC`. However, the same pilot may have chosen the `AircraftA` direction in the same situation. A strategy/behavior for the `AirCraftAPilot` for this situation will have to provide both these action options to the `AirCraftAPilot`. In order to be able to provide such a semantics, we first define nondeterministic strategies. In the discussion below, for the sake of simplicity, we use NCGS as our ongoing model at first to present some concepts.

6.9.1 Nondeterministic Strategies

Definition Let us consider an NCGS $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta \rangle$. A strategy for a player $p \in P$ is a function $s : Q^+ \to 2^\Sigma \setminus \{\}$, where for each $\lambda \in Q^+$ we must have $s(\lambda) \subseteq e_p(\lambda_{|\lambda|})$.

Given a state $q \in Q$, a set of players $\mathcal{P} \subseteq P$, and a set $S_\mathcal{P} = \{s_p | p \in \mathcal{P}\}$ of nondeterministic strategies, we define the outcomes of following the strategies in $S_\mathcal{P}$ from a state q to be the set $O(q, \mathcal{P}, S_\mathcal{P})$ of q-computations such that a q-computation $\lambda = q_0, q_1, \ldots \in O(q, \mathcal{P}, S_\mathcal{P})$ if $q_0 = q$ and for all positions $i \geq 0$, there is an action vector $\bar{a} \in \prod_{p \in P} e_p(q_i)$ such that (i) $\bar{a}_{|p} \in s_p(\lambda_0^i)$ and (ii) $q_{i+1} \in \delta(q_i, \bar{a})$. A nondeterministic strategy S_p is a deterministic strategy for the player p if $\forall \lambda \in Q^+.\ |S_p(\lambda)| = 1$. Every deterministic strategy is a nondeterministic strategy. Deterministic strategies are the same as strategies presented in [9]. In order to determine how nondeterministic strategies are related to deterministic strategies we first define strategy refinement and then state the relationship between the outcomes of following two strategies where one is a refinement of the other.

Definition Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be an NCGS, $\mathcal{P}, \mathcal{P}' \subseteq P$ be two sets of players where $\mathcal{P} \subseteq \mathcal{P}'$. Let $S_\mathcal{P}$ and $S'_{\mathcal{P}'}$ be sets of strategies for the players in $\mathcal{P}$ and $\mathcal{P}'$. Then $S_\mathcal{P}$ is a refinement or substrategy of $S'_{\mathcal{P}'}$ w.r.t. the set of players $\mathcal{P}_s \subseteq \mathcal{P}$ denoted by $S_\mathcal{P} \sqsubseteq_{\mathcal{P}_s} S'_{\mathcal{P}'}$ if we have that $\forall p \in \mathcal{P}.\ \forall \lambda \in Q^+.$ if $p \in \mathcal{P}_s$ then $S_p(\lambda) \subseteq S'_p(\lambda)$ else $S_p(\lambda) = S'_p(\lambda)$.

Lemma 6.9.1 *Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be an NCGS. Let $\mathcal{P} \subseteq P$ be a set of players in C and let $S_\mathcal{P}$ and $S'_\mathcal{P}$ be two sets of nondeterministic strategies where $S_\mathcal{P} \sqsubseteq_\mathcal{P} S'_\mathcal{P}$. Then $\forall q \in Q.\ O(q, C, S_\mathcal{P}) \subseteq O(q, C, S'_\mathcal{P})$.*

Proof Let us consider a state $q \in Q$. Let r be a q-computation such that $r \in O(q, C, S_\mathcal{P})$. For all $i \geq 0$, there exists an action vector $\bar{a}$ such that $r_{i+1} \in \delta(r_i, \bar{a})$ and $\forall p \in \mathcal{P}.\ \bar{a}|_p \in S_p(r_0^i)$. Then by Definition 6.9.1 we have $\forall p \in \mathcal{P}. \bar{a}|_p \in S'_p(r_0^i)$. Hence we can say $r \in O(q, C, S'_\mathcal{P})$.

6.9.2 Strategic Semantics of ATL* with NCGS without Strategy Refinement

In the original semantics of ATL* [9], whenever a player restriction is reached all responsibility of satisfying the rest of the formula is delegated to those players and they can chose a new course of action even if they had been proceeding with a different plan beforehand. The strategy for a group of players $\mathcal{P}$ under consideration should not be revoked if another coalition of players are trying to formulate a strategy to make parts of φ to be true. Instead the strategy under consideration should be consulted at every step a strategy for a player in $\mathcal{P}$ is needed. We now present a semantics for ATL* which deviates from this approach in three ways: (i) it handles nondeterministic strategies, (ii) it provides an understanding of whether a strategy for a set of players are sufficient for satisfying an ATL* formula, (iii) it affixes strategies to players: once a strategy is chosen for a player the strategy can not be revoked or modified during the rest of the analysis. Given an NCGS C, a state q, a set of players $\mathcal{P}$, a set of nondeterministic strategies $S_{\mathcal{P}}$ containing one strategy for each player in $\mathcal{P}$, a state q and an ATL* formula φ, we now present strategic semantics for ATL*:

- $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I v \in \Pi$ if $v \in \pi(q)$.
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \neg\varphi$ if $\mathcal{P}, C, q, S_{\mathcal{P}} \not\models_I \varphi$
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \varphi_1 \vee \varphi_2$ if $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \varphi_1$ or $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \varphi_2$
- $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \langle\!\langle \mathcal{P}' \rangle\!\rangle \psi$ if there exists a set of strategies $S_{\mathcal{P}' \setminus \mathcal{P}}$ such that $\forall \lambda \in O(q, C, S_{\mathcal{P}} \cup S_{\mathcal{P}'}).\ \mathcal{P} \cup \mathcal{P}', C, \lambda, S_{\mathcal{P}} \cup S_{\mathcal{P}'} \models_I \psi$
- $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \psi$ if $\mathcal{P}, C, \lambda_0, S_{\mathcal{P}} \models_I \psi$
- $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \neg\psi$ if $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \not\models_I \psi$
- $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \psi_1 \vee \psi_2$ if $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \psi_1$ or $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \psi_2$
- $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \bigcirc\psi$ if $\mathcal{P}, C, \lambda_1^{\infty}, S_{\mathcal{P}} \models_I \psi$
- $\mathcal{P}, C, \lambda, S_{\mathcal{P}} \models_I \psi_1 \mathcal{U} \psi_2$ if $\exists j \geq 0.\ \forall 0 \leq i < j.\ \mathcal{P}, C, \lambda_i^{\infty}, S_{\mathcal{P}} \models_I \psi_1$ and $\mathcal{P}, C, \lambda_j^{\infty}, S_{\mathcal{P}} \models_I \psi_2$

Now that we have nondeterministic strategies and strategic semantics in place we can relate the two concepts.

Theorem 6.9.2 *Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be an NCGS. Let $\mathcal{P} \subseteq P$ be a set of players and let $S_{\mathcal{P}}$ be a set of nondeterministic strategies and $q \in Q$ be a state. Then $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \varphi$ iff $\forall S'_{\mathcal{P}} \sqsubseteq_{\mathcal{P}} S_{\mathcal{P}}.\ \mathcal{P}, C, q, S'_{\mathcal{P}} \models_I \varphi$.*

Proof Given $\mathcal{P}, C, q, S_{\mathcal{P}} \models_I \varphi$. The most interesting case is again when $\varphi = \langle\!\langle \mathcal{P}' \rangle\!\rangle \psi$. Then by definition we have that there exists a set of strategies $S_{\mathcal{P}' \setminus \mathcal{P}}$ for players in $\mathcal{P}' \setminus \mathcal{P}$ such that $\forall \lambda \in O(q, C, S_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}}).\ \mathcal{P} \cup \mathcal{P}', C, \lambda, S_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}} \models_I \psi$. Now let us consider a nondeterministic strategy $S'_{\mathcal{P}}$ such that $S'_{\mathcal{P}} \sqsubseteq_{\mathcal{P}} S_{\mathcal{P}}$. Then we have $S'_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}} \sqsubseteq_{\mathcal{P}} S_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}}$ by Definition 6.9.1. Then from Lemma 6.9.1 we know that $O(q, \mathcal{P}, S'_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}}) \subseteq O(q, \mathcal{P}, S_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}})$. Combining all these facts we get, $\forall \lambda \in O(q, \mathcal{P}, S'_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}}).\ \mathcal{P} \cup \mathcal{P}', C, \lambda, S'_{\mathcal{P}} \cup S_{\mathcal{P}' \setminus \mathcal{P}} \models_I \psi$. Hence, $\mathcal{P}, C, q, S'_{\mathcal{P}} \models_I \langle\!\langle \mathcal{P}' \rangle\!\rangle \psi$. Since we proved it for an arbitrary subset of $S_{\mathcal{P}}$, this result will hold for all possible substrategies of $S_{\mathcal{P}}$. ▮

Corollary 6.9.3 *Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be an NCGS. Let $S^N_{\mathcal{P}}$ be a set of nondeterministic strategies for a set of players $\mathcal{P}$. Then $\mathcal{P}, C, q, S^N_{\mathcal{P}} \models_I \varphi$ iff there exists a set of deterministic strategies $S^D_{\mathcal{P}}$ such that $\mathcal{P}, C, q, S^D_{\mathcal{P}} \models_I \varphi$.*

Proof (Sketch) Let us construct the deterministic strategies $S^D_{\mathcal{P}}$: for each $\lambda \in Q^+$ and for each $p \in \mathcal{P}$, we set $S^D_p(\lambda) = \{a\}$ where $a \in S^N_p(\lambda)$. Then by Definition 6.9.1 we have $S^D_{\mathcal{P}} \sqsubseteq_{\mathcal{P}} S^N_{\mathcal{P}}$. Now from Theorem 6.9.2 we have $\mathcal{P}, C, q, S^D_{\mathcal{P}} \models_I \varphi$. For proof in the other direction, we observe that every deterministic strategy is a nondeterministic strategy. Then if we have $\mathcal{P}, C, q, S^D_{\mathcal{P}} \models_I \varphi$ then there exists a set of nondeterministic strategies $S^N_{\mathcal{P}}$ such that $\mathcal{P}, C, q, S^N_{\mathcal{P}} \models_I \varphi$ where $S^N_{\mathcal{P}} = S^D_{\mathcal{P}}$.

Ågotnes *et al* proposes modeling irrevocable strategies and presents MIATL (Irrevocable strategy ATL with memory) in [7]. They perform model update every time a strategy is fixed for a player. Model updates are essentially restricting the *e* function according to a strategy. Since their strategies are deterministic ones the *e* function cannot be further refined even if the player again gets a chance to choose a strategy. They first provide a tree unfolding of CGSs and then restrict the tree-like unfolded CGS according to the strategy under consideration. Our semantics carries the strategies around instead of updating the models with them. We now show how we can relate the two different types of semantics.

Theorem 6.9.4 *Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta \rangle$ be an NCGS and ϕ be an ATL* formula. Then for a state $q \in Q$, for a set of players $\mathcal{P} \subseteq P$, $\{\}, C, q, e \models_I \varphi$ iff $C, q \models_{MIATL*} \varphi$.*

Proof (Sketch) If $\{\}, C, q, e \models_I \varphi$ then we can show that $C, q \models_{MIATL*} \varphi$ by induction on the structure of φ. The only interesting case is when $\varphi = \langle\!\langle \mathcal{P} \rangle\!\rangle \psi$. At that point a nondeterministic strategy is affixed to every player in $\mathcal{P}$. Then by Corollary 6.9.3, there are deterministic strategies that can be affixed to these players. From now on players in $\mathcal{P}$ will behave according to those strategies. In [7], C (or rather its tree unfolding) is updated with some deterministic strategies for $\mathcal{P}$. We can use the strategies chosen in our semantics as witness strategies used by their semantics. Then the outcomes in our semantics will be the same as outcomes in their updated model. Proof for the opposite direction is similar.

6.9.3 Model Checking Complexity

In [7], the model checking complexity for IATL (irrevocable strategy ATL) is determined by showing a translation of deterministic CGSs to model checking ATL$_{ir}$ of [103]. Following this path we determine complexity of model checking ATL* with irrevocable strategies with NCGSs. First we translate the NCGS C to a deterministic CGS D. Then we consider a tree unfolding [7] of the DCGS. Since we start with a specific set of strategies we update the tree model with the those strategies first and then perform MIATL model checking on that updated tree model. Since MIATL model checking on a CGS equivalent to IATL model checking on a tree unfolding of the CGS, MIATL model checking is of the same complexity as IATL model checking [7]. So model checking ATL* with irrevocable strategies with NCGSs is in NP wrt the size of the model and the formula.

6.9.4 Strategic Semantics of ATL* with NCGS with Strategy Refinement

The previous semantics does not allow any modification of a strategy chosen for a player. We now provide a strategic semantics which is more flexible in the sense that each new player coalition will allow the players to refine the strategies assigned to them. They will still have to operate within the constraints set by the initial strategy but they will have the option to further restrict their choices each time a player coalition involving them is considered. This semantics is presented below with the help of notions of strategy refinements. Given a a set of players $\mathcal{P}$, a set of strategies $S_{\mathcal{P}}$ containing one strategy for each player in $\mathcal{P}$, a state q and an ATL* formula φ, we present strategic semantics with strategy refinements as follows:

- $\mathcal{P}, C, q, S_{\mathcal{P}} \models_R \langle\langle \mathcal{P}' \rangle\rangle \psi$ if there exists a set of strategies $S_{\mathcal{P}'\setminus\mathcal{P}}$ a set of strategies $S'_{\mathcal{P}}$ where $S'_{\mathcal{P}} \sqsubseteq_{\mathcal{P}} S_{\mathcal{P}}$ such that $\forall \lambda \in O(q, C, S'_{\mathcal{P}} \cup S_{\mathcal{P}'\setminus\mathcal{P}}).\mathcal{P} \cup \mathcal{P}', C, \lambda, S_{\mathcal{P}} \cup S_{\mathcal{P}'\setminus\mathcal{P}} \models_R \psi$

Since this semantics relies on the existence of a substrategy for $S_{\mathcal{P}}$ that will be sufficient to maintain an ATL* guarantee, we can relate $S_{\mathcal{P}}$ with its substrategies only in the following manner.

Theorem 6.9.5 *Let $C = \langle P, Q, Q_0, \Sigma, \Pi, \pi, e, \delta\rangle$ be an NCGS. Let $\mathcal{P} \subseteq P$ be a set of players and let $S_{\mathcal{P}}$ be a set of nondeterministic strategies and $q \in Q$ be a state. If $\mathcal{P}, C, q, S_{\mathcal{P}} \models \varphi$ then $\exists S'_{\mathcal{P}} \sqsubseteq_{\mathcal{P}} S_{\mathcal{P}}$ such that $\mathcal{P}, C, q, S'_{\mathcal{P}} \models \varphi$.*

Proof Given $\mathcal{P}, C, q, S_{\mathcal{P}} \models \varphi$. The most interesting case is again when $\varphi = \langle\langle \mathcal{P}' \rangle\rangle \psi$. Then by definition we have that there exists a set of strategies $S_{\mathcal{P}'\setminus\mathcal{P}}$ for players in $\mathcal{P}' \setminus \mathcal{P}$ and a substrategy $S''_{\mathcal{P}}$ such that $\forall \lambda \in O(q, C, S''_{\mathcal{P}} \cup S_{\mathcal{P}'\setminus\mathcal{P}})$. $\mathcal{P} \cup \mathcal{P}', C, \lambda, S''_{\mathcal{P}} \cup S_{\mathcal{P}'\setminus\mathcal{P}} \models \psi$. From these facts we can say $\mathcal{P}, C, q, S'_{\mathcal{P}} \models \langle\langle \mathcal{P}' \rangle\rangle \psi$ because $S'_{\mathcal{P}}$ is a refinement of itself. ∎

6.10 Summary

We have presented iNCGSs capable of handling incomplete information and nondeterminism. We have presented ATL and ATL* semantics with respect to iNCGS. We have presented a symbolic model checking algorithm for model checking ATL with respect to iNCGS. We also demonstrate how iNCGSs can relate to other existing multi-agent game structures with incomplete information like iTSGS and game arenas. We have presented strategies as an improved modeling formalism for human behavior. We provide semantics for ATL and ATL* where we judge whether a human behavior modeled using a strategy can satisfy a property.

Chapter 7

Related Work

7.1 Human Task Execution Analysis

7.1.1 Operator Task Analysis

The term "Operator Task Analysis" was first proposed by Lees in [72]. This work, and others with similar objectives [52, 67], aims to determine how a task is actually accomplished by human operators, what special factors are involved in or required of the operators to accomplish the goal the task is supposed to achieve. This provides insight into how human operators interact with the machines and possibly indicates how the machinery can be improved to simplify the operator task. A key goal is to analyze the possible operator action variances from a specified set of tasks depending upon environmental effects. Hierarchical Task Analysis (HTA) performs similar tasks by decomposing complex tasks in a hierarchical manner [42]. Action Error Analysis (AEA) [112, 113] attempts to find the possible future deviations in operator behavior. Work Safety Analysis (WSA) [112, 122] analyzes each step in a specific task and analyze how variations in performing that step can cause different types of hazards. Tasks have been formally modeled using graphical techniques like fault trees [74], cause-sequence diagrams, Petri nets and logical techniques like real time logic [61] and predicate logic [62]. Operator Function Model (OFM) [85] provides a finite state machine based analytical tool to give a task analytic structure of operator behavior. However OFM lacks ability to efficiently model and analyze concurrent system parts. Object-Z [25] is an object-oriented task specification language that formally models operator tasks. Although it models some operator action aberrations like repetition and omission, it does not model concurrency.

7.1.2 Human Reliability Analysis

Another closely related field of study is Human Reliability Analysis (HRA) which mostly performs probabilistic study of possible influence of environment on humans to find variations in human operator actions. The goal of HRA is mostly focused on determining the actions in the protection envelope so that the system can be extended or designed to be able to tolerate such actions. First generation HRA techniques mostly computed the probability that a human operator will make a certain kind of error. Second generation HRA techniques like HEART [124], MERMOS [20], CREAM [57], ATHEANA [35] attempt to improve the first generation HRA techniques by performing human error analysis at different levels of system implementation like design, creation, and operation stages. This work also considers the effect of various contexts on human behavior while interacting with computer systems.

7.1.3 Workflow

Apart from HRA techniques, there is an emerging research trend for specifying, modeling and verifying human workflow using the currently available workflow specification techniques often originally aimed at modeling business processes implemented as web services. The most prominent workflow specification techniques are Yet Another Workflow Language (YAWL) [120], and the Business Process Execution Language (BPEL) [10]. UML [45] also provides a graphical method for similar applications. BPEL is (arguably) the most widely accepted technique by the business community. Both BPEL and YAWL provide tools for modeling workflow. However, generic workflow languages treat human operators as any other elements of the system. Hence a need to specialize workflow specification languages to support the unique human operators is a new area of attention. BPEL has been extended for human task specification to give BPEL4People [6] which is capable of describing distinguished tasks to be performed by specific group of people. The YAWL system uniformly supports several types of human task allocation and handling. Collaborative Activity Human Workflow (CAHW) [115] provides support to specify human workflow and activities and collaboration among them. These human workflow specification techniques are only aimed at specifying recommended workflow for human operators and do not provide support for describing or reasoning about operator variances.

Workflow Verification

Typically workflow verification entails determining whether a workflow is deadlock free, live, livelock free, conforms to the desired goals and so on. Many different formal models have been researched on their usability as the model for workflow analysis. Many of the major process algebra formalisms have been used for this purpose including studies based on CSP [125], Communicating Concurrent Systems (CCS) [83, 111], and the π-Calculus [84, 95]. Moreover, BPEL4People has been formally modeled using CSP [129]. Among non-process algebraic models, the most prominent is that of Petri nets [90, 118]. Petri nets have both graphical appearance suitable for workflow modeling and mathematical model for analysis. They can be used to model YAWL, which, in turn, can model a wide range of workflow patterns [121, 5, 118]. WorkFlow net or WF-net [119] is an extension of Petri nets for expressing business processes. We used CSP to model and analyze operator workflow in our hospital AIDC project [51]. The main correctness properties there were proved by hand, but we also tried to prove them with formal tools that support CSP. We first used

FDR2 (`fsel.com`) a model checker for CSP process to try to verify some refinement theorems for the model of our system. Then we used an interactive theorem prover for CSP processes namely the CSP-Prover [60] to attempt to prove the same refinement theorems. CSP-Prover is built on top of the Isabelle theorem prover (`isabelle.in.tum.de`).

In another direction in the formal analysis of workflows, LTL-WF [127] combines WF-nets and linear temporal logic so that expressing temporal properties for workflows becomes easier. Logic (CTR) [40] is used to specify, analyze, and schedule workflows. YAWL has a model checker incorporated in its available software (`yawl-system.com`) for verifying some aspects of a workflow expressed using YAWL. The YAWL model checker models YAWL workflows using reset nets [11, 44], a generalized version of Petri nets capable of handling multiple tokens at a time.

7.1.4 Web Services Orchestration

Web services orchestration [89] provides a standards-based approach for allowing web services to interact together to create a higher level business process. Web service choreography is more collaborative in nature in the sense that it tries to track and manage public interactions among multiple web services. Orchestration and choreography standards are required to ensure that the underlying infrastructure can handle workflow variations like errors and time outs and to allow for compensating transactions. Compensating transactions are tasks performed to undo a transaction without causing dependent or related transactions to abort. Compensation techniques [48] seek to semantically remove the effects of a transaction in a manner that preserves the consistency of the overall system. Compensation is essentially a way to create a protection envelope, but compensation techniques in the current literature are oriented toward database transactions rather than operator error.

7.1.5 Fault-Tolerant System

Fault-tolerant or fail-safe system design is another related field of work [44]. Fault-tolerant system aim to design systems that can continue executing even when parts of the system fail. Byzantine fault tolerance techniques [71, 28] aim to sustain overall system functionality even when some components are not only behaving erroneously but also acting inconsistently while interacting with multiple other components. Fault-tolerant system design is devoted to first determining which parts of a hardware system, or in some cases software [77], may fail, how critical its failure will be, what measures (*viz.* redundancy) can be taken to ensure that in case of such a failure enough backup measures are present to mitigate the fault and keep the overall system functioning. So, while our goal is to focus on the actions of the users of the system causing a possible error, their focus is on the computer system itself failing to function properly. Another noteworthy work is the simplex reference model for fault-tolerant systems as given in [36] which comprises of four components: the external environment influencing the operation of the system, knowledge about its external environment, the machine control part which should be fault tolerant through redundancy, and the safety requirements that should be maintained to guarantee a certain level of operation in the face of faults occurring in the system or its use. The underlying idea of this work is similar to our goal, but our focus here is on modeling and analyzing the human component of the external environment and how its behavior variance affects the safety requirements.

7.1.6 Formal Models for Human Actions

Lindsay *et al* use CSP to model operator behaviors for an air-traffic control scenario in [76]. They use temporal logic assertions to formally analyze human errors in that setting. Their focus is more on the cognitive aspects of human errors. It is also not clear from their work how they realize their idea of model checking CSP process traces against temporal logic properties. Another sequence of works on human cognitive process modeling is that of Curzon *et al* [37, 39, 101, 38]. They formally model and verify the human cognitive process: how a human interprets the interface provided to him by a system, what actions he may perform based on his interpretations. They mostly model the outcomes of human cognitive process as nondeterministic choice between all possible correct and erroneous decisions. This is the same as our approach. Although we focus not on the human decision process but on the actual decision, our nondeterministic choice model for decision among possible actions is the same as their approach. Our technique implicitly models human cognitive process.

In [105] Shin *et al* use graphs and deterministic finite state automata to model and analyze human operator behavior. However, their approach is limited to recommended behavior analysis. Clarke *et al* show in [33] how medical processes or any human-intensive processes can be specified using a process specification language Little-JIL, and then subsequently translated to finite state automata and analyzed for satisfaction of desired properties. Their focus is on providing a collaborative task decomposition for humans to follow. At each level of decomposition, Little-JIL also allows modeling the actions recommended to the humans to take if an exceptional situation arises. For example, a customer goes to purchase a train ticket but finds out all tickets have been sold. Our focus is different from their work in that we only consider basic recommended actions and do not consider advices for exceptional situations. We are more concerned about how the humans themselves can cause exceptions instead of how they should handle the exceptional situations. We determine whether the system is robust against exceptional situations caused by the humans. However, our recommended task descriptions are very simple sequences and handling hierarchical description of a recommended task is a future direction. There is another aspect in which our methodology is different from their methodology. In their work, primary focus is on human task decomposition modeling. The computer system is not modeled using Little-JIL. It is encoded in the language of the model checker they use later on. We however take an uniform approach towards encoding humans and systems. For example CSP can be used to succinctly model both human and system behaviors.

Bolton *et al* extend OFM to EOFM [23] with task sequencing and conditional constraints to better model human task behavior. EOFM provides a formal model for hierarchical task decomposition for humans. However, EOFM does not provide elegant mechanism for modeling multiple human operators, their interactions among themselves, interactions among the humans and the system and possibly other environment components. Nor does it provide elegant modeling of parallel possibly synchronized executions of these aforementioned entities. In fact like Little-JIL work, the system is ignored until actual property verification is performed using a model checker. One of their work with EOFM [24] where they use the phenotypes of erroneous actions of Hollnagel to predict possible source of errors for a system is very close to us. Given a recommended task EOFM for a human operator their approach is to (i) create another EOFM model which has all possible error phenotypes embedded in it, (ii) model check the new EOFM against system requirements to find a counter example. This counter example will demonstrate how an erroneous action might be problematic for a system. Our approach is different in that at one time we only introduce a single phenotype. Then we present which phenotypes the system is robust against and which phenotypes the system is *not* robust against. Thus the system designers will now have a thorough analysis of all possible erroneous actions and their possible effects on the system.

7.1.7 Model Checking

The usual trend in formal analysis in determining whether a system maintains a guarantee is by performing model checking [32]. Model checking a system M is determining whether M satisfies a requirement ψ denoted by $M \models \psi$. Although this works well for "closed" systems (systems with no interaction with the rest of the world), for "open" (systems that interact with their environment) systems it is not sufficient. This has given rise to "module checking"[69] where a . This has given rise to the notion of "robust satisfaction" of a property [70]: a system M should satisfy a property ψ when composed with any environment E, $M \| E \models \psi$. This trend is the closest to our work here. Any human operator system should be verified against all possible reasonable human behaviors. Our goal is to provide a framework to assist in formulating reasonable or protected human task execution behavior and determining that they are indeed protected behavior. However as suggested in [70], checking against all possible is not practical. Hence we focus on reasonable human behaviors.

7.1.8 Fuzz Testing

Fuzz testing or fuzzing involves testing a computer system against unexpected, erroneous, random inputs [82]. A software is provided with a *fuzzed* input and monitored against hazardous consequences like crashes or invariant violations. Our technique of testing robustness of a system against erroneous human actions is very similar to fuzz testing. Given a recommended task specification we systematically introduce erroneous actions patterns into it. Then we check the system for safety or protection envelope property conformation.

7.1.9 Controller Synthesis

Given a definition of reasonableness of behavior we will build a model for all models conforming to that definition. Controller synthesis works are related in that regard. Controller synthesis [92] works model the interactions between a system and its environment as a discrete game. Controller synthesis aims at generating a winning strategy for the controller such that the system always wins irrespective of the behavior of the environment.

7.1.10 Interface Synthesis

Interface synthesis research [55, 107] is a related research area. In modular program analysis, individual modules can be analyzed separately and then only interfaces are used when the overall system composed of all relevant modules are handled. Interfaces summarize acceptable call sequences for a module. An interface is safe if it only allows sequences that do not violate the internal invariants of its module. Permissive interfaces are those that allow all possible safe sequences. Full interfaces are both safe and permissive. Safe interfaces correspond to the protected human behaviors as presented in the next subsection. Full interfaces are achieved when the all safe behaviors are protected behaviors. Given a regular language description of an interface and its invariants, interface synthesis techniques provide models that accept words belonging to that language. Given the behavior guarantees expected from the human operators, we will use refinement to derive a model representing all well behaved human action sequences. Interface synthesis is performed for software modules for languages like Java also [8]. Software environment generation from environment assumptions is also a related to our work [116]. The concepts of these works are very related to our work. Interface Automata [41] is also related. These works focus on modular software or hardware systems. For one system module, the environment is comprised of other modules from that system. However, the primary novelty of our work is the context chosen by us: human operators that will interact with a system composed of hardware and software, concepts of categorizations of their interaction behaviors, guarantees systems can provide based on different categories of assumptions about human operators. However, the underlying concepts used in our work is different from them. If we consider human operators as modules then we see that they can have different safety invariants and expected behavior guarantees. In the self-scan checkout scenario, a human module is safe for the human customer if he is not made to pay for items he did not purchase. However, the customer needs to ensure that he will not take items out of the store without paying for them first. Since the supermarkets cannot distinguish items that have not been paid for, if the customer does not guarantee conforming to the required manner of operation, the supermarket can suffer loss. For human operators we focus on the guarantees that they need to provide to ensure safety of the automated system, for the automated system we focus on their safety based on the guarantees maintained by human operators. Identifying behavior requirements and individual safety requirements and then devising and analyzing models for them is the main focus of our work.

7.1.11 Rely-Guarantee

The rely-guarantee method of software verification was originally devised for verification of multi-threaded programs [63]. A thread provides a guarantee and the rest of the threads will rely on this guarantee to ensure safe operation. This idea generalizes to modular software verification. Often it is easier to verify each software module individually and later on combining the results to determine overall safety preservation. Each module will provide a guarantee where the module has to rely on guarantees provided by the other modules to provide this guarantee on its part. If there is clear notion about the guarantees each module needs to provide then the modules can be verified individually. The concept of rely-guarantee is very close to our concept of protection envelope. Each entity irrespective of whether it is a software component or a human operator needs to provide a guarantee of always staying within their protection envelopes.

7.2 Imperfect Information

ATL and CGS were presented by Alur *et al* in [9]. They presented ATL, ATL*, Alternating μ-calculus, Fair ATL, Fair ATL* and ATL with incomplete information along with their model checking complexities. Incomplete information game structures have been widely researched [78]. Reif [98] has shown that incomplete information games are undecidable. Alur *et al* present iTSGS in [9] which they claim cannot be model checked based on the proofs given in [98]. We extend their work by introducing nondeterminism in the transition system of CGSs and allow the players to have incomplete information about the state space and state sequences. Ågotnes *et al* presented ATL with irrevocable strategies, IATL and MIATL in [7]. IATL only considers memoryless strategies and MIATL considers strategies that can consider a sequence of states to provide guidance. We provide strategic semantics for ATL formula using NCGS which provides a mechanism to permanently associate a set of strategies with a set of players and thereby mimic irrevocable strategies. Schobbens present ATL with imperfect information modeled using imperfect information CGS or iCGS in [103]. He proposes four different types of strategies based on their having (in)complete view of the set of states and being able to consider history (or not). Our notion of state and trace equivalences subsume their work. Another related work is that of Brihaye *et al* who present ATL with strategy contexts and bounded memory $\text{ATL}^*_{sc,\infty}$ and some variations of it in [26]. They analyze expressive power and model checking complexity for their logics. Their semantics is very related to the strategic semantics proposed by us in Section 6.9. The strategies carried around in our semantics is similar to their strategy contexts. Dima *et al* present a very related work in [43]. They provide a model checking algorithm for ATL with knowledge, imperfect information, prefect recall and communicating coalitions and show that their version of ATL model checking with incomplete information is not undecidable.

Broersen *et al* augment ATL with STIT(*S*eeing *T*o *I*t *T*hat) operators and present strategic semantics for Alternating Transition Systems (ATS) in [27]. They also extend ATL with epistemic operators in [27]. Mogavero *et al* present relentful strategic reasoning for ATL* and mATL* in [86]. Usually in ATL semantics the history is thrown out when a new player coalition is reached. They suggest "memoryful" traces where each path is considered starting from its past not from the present. They present variations of ATL* and mATL* namely pATL*, mpATL*, m^-ATL* and p^-ATL* along with comparisons of their expressive power and satisfiability and model checking complexity. Here *p* stands for past time and these logics are augmented with "previous" and "since" operators.

7.3 Runtime Verification

Runtime verification of a software performs dynamic analysis of a program. The interesting aspect of runtime analysis is that in this case the actual software can be analyzed instead of an abstract model. However, runtime verification analyzes one specific execution trace at a time. Thus runtime analysis falls in between static verification and unit testing. Runtime monitoring deploys a software monitor which continuously monitors an execution of a software to detect whether it violates a desired property. We use runtime monitoring techniques and software to study the possibility of runtime verification of a human operator behavior guiding the execution of a human-intensive safety-critical system. In this regard all runtime monitor work is related as background work [15, 18, 21, 17, 80, 81, 31, 12, 14, 18, 15, 49, 106, 100, 130, 128, 64, 91, 66, 22, 34]. Psaier *et al* present a methodology for monitoring and self-adaptation approach of service-oriented collaboration networks in [94]. For collaboration networks where some services are provided by human operators they provide a method for modeling and simulating human behavior through Web-services test-bed. The test-bed allows for simulating erroneous behavior of services. They use standardize Service-Oriented Architecture (SOA) infrastructures and they combine the capabilities of human and software services through the use of well-established standards like Simple Object Access Protocol (SOAP) and the Web Service Description Language (WSDL). The focus of their architecture is on developing monitors to monitor network services with capability for adapting the system is faults are detected. The focus on our work on monitoring was on presenting a generic framework for using existing monitoring techniques and softwares to study the effect of the human operator on a generic system.

Chapter 8

Conclusion

We have learned and demonstrated a number of things in this dissertation. We set out with a goal of providing methodologies for modeling and analyzing the effect of the task execution behaviors of human operators on requirement conformation of computer systems. We now summarize our achievements in this regard:

We have demonstrated that human operator behaviors manifested during execution of a task involving use of a computer system can be formally modeled and analyzed. The goal of such analysis is determining if safety requirements can be maintained by a computer system while being operated by human operators.

We have learned that the human operators demonstrate many different task execution behaviors while operating a computer system. The recommended task executions are the desired human operator behaviors. They are also the easiest to formulate and analyze. Verification of a computer system while combined with human operators operating them in the recommended manner provides a necessary sanity check for recommended task specifications.

We have demonstrated that both formal verification techniques like process refinement checking, model checking and runtime verification can be used for assessing the resilience of a computer system against human operator behaviors. We have shown how human behavior can be modeled in well established software programming platforms and then can be dynamically monitored for safety conformation for a computer system. The technique suggested is robust in the sense that just as we can analyze at runtime with a model of a human operator, we can use runtime monitoring to monitor the executions of the computer system itself while being operated by real life operators.

We have shown that each computer system would benefit from a precise formulation of the guarantees of *good* or tolerable behavior that it needs to rely upon to provide requirement conformation. These tolerable behavior guarantee or protection envelopes provide an understanding of the shortcomings of the computer systems in being able to meet safety requirements. If the human operators do not restrain themselves to operate within the protection envelopes, then the system cannot guarantee safety requirement conformation.

We have demonstrated that we can formally specify and analyze these protection envelope properties for each human operator of a computer system. We have also demonstrated that we can study the robustness of these protection envelopes. We can formally assess if the computer system is resilient against well established patterns of atypical task execution behaviors exhibited by human operators.

We have also demonstrated that in the presence of multiple human operators, we can determine the human operator responsible for violation of safety property. We perform this analysis by combining formal analysis with the concept of protection envelope. Analysis of the system and human operators against protection envelope properties help us identify the human operator whose protection envelope got violated. The human operator whose protection envelope gets violated did not maintain his guarantee of *tolerable* behavior and is a possible reason behind safety violation of the system.

We have demonstrated that Concurrent Game Structures (CGS), the formalism chosen by us for modeling deviant human operator behaviors was lacking in some aspects for our purposes. CGS provide a procedural method of encoding the concurrent operation of humans and computer systems. System safety requirements to be imposed upon the structure declaratively. The elements of the structures can be conveniently manipulated to obtain representative erroneous human behavior models. However, they lack nondeterminism and they do not support information hiding among agents. We provide extension of CGSs with these features and extend the semantics for Alternating Time Temporal Logic (ATL) for this setting. We also provide a strategy based semantics for ATL which is a more faithful representation of a human task execution behavior satisfying a property.

In a nutshell, we have demonstrated that expected and erroneous human operator task execution behaviors can be formally encoded and analyzed thereby providing a formal understanding of the resilience of a computer system against both desired and atypical human operator behaviors.

In future, we plan to further carry on our research on broadening the work on modeling human operators and analyzing them. We have the opportunities of increasing the level of automation at every step of the frameworks we have presented: from the formulations of protection envelopes to building abstract models for computer systems and human operators. Also in some parts of our work like analysis for determining robustness against erroneous human actions we allowed for only simple sequential description of human tasks. A future direction is to consider more complex task description allowing for decision procedures and possibly exceptional situation handling advice. Runtime verification work needs to be furthered so that we can provide a low-overhead generic method of verifying the actual computer system against erroneous human operator actions. We will further our work on the modeling of human operator behaviors through player strategies in CGSs. We intend to explore complex, multi-layer protection envelopes for collections of human operators. To facilitate this we intend to explore extensions for ATL* and model checking algorithms for them.

Appendix A

Nurse Behavior and Mobile Mediator Reaction in AIDC Scenario

```
package aidcwithaspect;

public class Main {

    public static void main(String[] args) {
        Nurse nurse1 = new Nurse(173);
        EHR backEndEHR = new EHR();
        MobileMediator mmed = new MobileMediator(backEndEHR,456);

        RFIDTag tag1 = new RFIDTag(1);
        RFIDTag tag2 = new RFIDTag(2);
        RFIDTag tag3 = new RFIDTag(3);

        Patient patient1 = new Patient.Builder(tag1).bloodPressure(130).
                                        pulseOxygenLevel(22).build();
        Patient patient2 = new Patient.Builder(tag2).bloodPressure(105).
                                        pulseOxygenLevel(29).build();

        Device device = new Device(tag3);

        nurse1.assignMobileMediator(mmed);
        nurse1.scanPatientIDwithMobiMed(patient1, mmed);
        nurse1.scanDeviceIDwithMobiMed(device, mmed);

        nurse1.readScannedDeviceID(mmed);
        nurse1.readScannedPatientID(mmed);
        mmed.getApprovalForScannedIDs(nurse1.approveScannedIDs(mmed));
        nurse1.captureDataFromDevice(mmed, patient2, device);
        nurse1.readCapturedDataFromMMed(mmed);

        mmed.getCapturedDataApproval(nurse1.approveDisplayedReading());
        }
}
```

Listing A.1: Human Operation Behavior

Appendix B

Nurse using AIDC System

```java
package aidcwithaspect;

/**
 *
 * @author Ayesha Yasmeen
 */
public class Nurse {
    private int nurseID;
    private int scannedPatientID = -1;
    private int scannedDeviceID = -1;
    private int observedData = -1;
    private int displayedReading = -1;
    private Patient currPatient ;
    private Device currDevice ;
    private int mmedID ;

    Nurse(int nurseid){
        nurseID = nurseid;
    }

    public int getNurseID(){
        return nurseID;
    }
    public void assignMobileMediator(MobileMediator mobileMediator){
        mmedID = mobileMediator.getMMedID();
        mobileMediator.associateWithNurse(this);
    }
    public void scanPatientIDwithMobiMed(Patient patient, MobileMediator

    mobileMediator){
        currPatient = patient;
        mobileMediator.captureRFIDTagFromPatient(patient);
    }
    public void scanDeviceIDwithMobiMed(Device device, MobileMediator

    mobileMediator){
        currDevice = device;
        mobileMediator.captureRFIDTagFromDevice(device);
    }

    public void readScannedPatientID(MobileMediator mobileMediator){

        scannedPatientID = mobileMediator.displayScannedPatientID();
    }

    public void readScannedDeviceID(MobileMediator mobileMediator){
        scannedDeviceID = mobileMediator.displayScannedDeviceID();
    }

    public boolean approveScannedIDs(MobileMediator mobileMediator){
        if (currPatient.getRFIDTag() == scannedPatientID)
            if(currDevice.getRFIDTag() == scannedDeviceID)
                return true;
        return false;
    }

    public void captureDataFromDevice(MobileMediator mobileMediator,
    Patient patient, Device device){
        observedData = mobileMediator.captureDataFromDevice(patient,
        device, true);
    }

    public void readCapturedDataFromMMed(MobileMediator mobileMediator){
        displayedReading = mobileMediator.displayCapturedDeviceData();
    }

    public boolean approveDisplayedReading(){
        if( displayedReading == observedData)
            return true;
        return false;

    }
```

```

}
```

Listing B.1: Human Operation Behavior

Appendix C

Monitor for AIDC Scenario

```
package mop;

import java.io.*;
import java.util.*;

AIDC(aidcwithaspect.Nurse nurse, aidcwithaspect.MobileMediator mmed,
aidcwithaspect.Patient patient, aidcwithaspect.Device device) {

int collectedData;
int patientID;
int patientID2;
int deviceID;
  event assignMMed after(aidcwithaspect.Nurse nurse,
        aidcwithaspect.MobileMediator mmed) :
    call(* aidcwithaspect.Nurse.assignMobileMediator(*))
    && target(nurse)
    && args(mmed)
  {}

  event nurseScansPatientID after(aidcwithaspect.Nurse nurse,
        aidcwithaspect.Patient patient, aidcwithaspect.MobileMediator mmed):
    call(* aidcwithaspect.Nurse.scanPatientIDwithMobiMed(*,*))
    && target(nurse)
    && args(patient, mmed)
  {
    patientID = patient.getRFIDTag();
  }

  event nurseScansDeviceID after(aidcwithaspect.Nurse nurse,
        aidcwithaspect.Device device, aidcwithaspect.MobileMediator mmed):
    call(* aidcwithaspect.Nurse.scanDeviceIDwithMobiMed(*,*))
  && target(nurse)
  && args(device, mmed)
{
  deviceID = device.getRFIDTag();
  System.out.println("Device getting used is: "+deviceID);
}
event nurseValidatesIDs after(aidcwithaspect.Nurse nurse,
  aidcwithaspect.MobileMediator mmed) :
  call(* aidcwithaspect.Nurse.approveScannedIDs(*))
  && target(nurse)
  && args(mmed)
{}
event nurseCapturesDataUsingDeviceAndMMed after(
      aidcwithaspect.Nurse nurse,
      aidcwithaspect.MobileMediator mmed, aidcwithaspect.Patient
      patient, aidcwithaspect.Device device):
  call(* aidcwithaspect.Nurse.captureDataFromDevice(*,*,*))
  && target(nurse)
  && args(mmed, patient, device)
{
  patientID2 = patient.getRFIDTag();
}
event mobimedCapturesReading after(
  aidcwithaspect.MobileMediator mmed,
  aidcwithaspect.Patient patient, aidcwithaspect.Device device,
  boolean b ) returning (int dataValue):
call(* aidcwithaspect.MobileMediator.captureDataFromDevice(*,*,*))
&& target(mmed)
&& args(patient, device,b)
{
  collectedData = dataValue;
  System.out.println("Collected\ "\+ collectedData);
}

event ehrStoresTupleWrongPatient after(aidcwithaspect.EHR ehr,
  aidcwithaspect.Nurse nurse, aidcwithaspect.MobileMediator
  mmed,aidcwithaspect.Patient patient, aidcwithaspect.Device
  device , int dataValue ):
call(* aidcwithaspect.EHR.storeTuple(*,*,*,*,*))
&& target(ehr)
&& args(nurse,mmed,patient,device,dataValue)
```

```
  && condition(patientID2 != patient.getRFIDTag())
  {
    System.out.println("Patient mismatch!
            Did not identify this patient");        }

  event ehrStoresTupleWrongDevice after(aidcwithaspect.EHR ehr,
    aidcwithaspect.Nurse nurse, aidcwithaspect.MobileMediator
    mmed, aidcwithaspect.Patient patient, aidcwithaspect.Device
     device , int dataValue ):
  call(* aidcwithaspect.EHR.storeTuple(*,*,*,*,*))
  && target(ehr)
  && args(nurse,mmed,patient,device,dataValue)
  && condition( deviceID != device.getRFIDTag())
  {
     System.out.println("Device mismatch! Did not
            identify this device");
  }

  event ehrStoresTupleWrongData after(aidcwithaspect.EHR ehr,
    aidcwithaspect.Nurse nurse, aidcwithaspect.MobileMediator mmed,
    aidcwithaspect.Patient patient, aidcwithaspect.Device device,
     int dataValue ):
  call(* aidcwithaspect.EHR.storeTuple(*,*,*,*,*))
  && target(ehr)
  && args(nurse,mmed,patient,device,dataValue)
  && condition( dataValue != collectedData)
  {
     System.out.println("Data mismatch! Did not collect this data");
  }

  ltl :  [] ( not ( ehrStoresTupleWrongDevice
               && ehrStoresTupleWrongPatient
               && ehrStoresTupleWrongData)  )

  @violation {
    System.out.println("Erroneous tuple
        getting inserted into EHR!");
  }
}
```

Listing C.1: Human Operation Monitor

Appendix D

CGSs for Self-scan Checkout Scenario Example

D.1 Sparse CGS

```
bool itembagged, startpressed, itemscanned, itempaid, erred, alarmraised;
mtype = { noaction, pressstart, scanitem, payforitem, bagitem, leavestore }

mtype = { noaction, allowpressstart, allowbag, allowpay, allowscan, errorReset }

mtype = { initial, pressedstart, scanneditem, baggeditem, paidforitem, error }
chan cschan = [1] of { mtype };
chan humchan = [1] of {mtype };
chan ctrlchan = [2] of {mtype};
proctype CS()
{
initialstate:
cschan!noaction;
if
:: humchan?noaction -> goto initialstate;
fi;
if
:: humchan?pressstart -> goto initialstate;
fi;
if
:: humchan?payforitem -> goto disabledHUMActionState;
fi;
cschan!allowpressstart;
if
:: humchan?noaction -> goto initialstate;
fi;
if
:: humchan?pressstart -> goto pressedstartstate;
fi;
```

```
pressedstartstate:
startpressed = true;
cschan!noaction;
if
:: humchan?noaction -> goto pressedstartstate;
fi;
if
:: humchan?pressstart -> goto disabledHUMActionState;
fi;
if
:: humchan?scanitem -> goto pressedstartstate;
fi;
cschan!allowscan;
if
:: humchan?noaction -> goto pressedstartstate;
fi;
if
:: humchan?scanitem -> goto scanneditemstate;
fi;
scanneditemstate:
startpressed = true;
itemscanned = true;
cschan!noaction;
if
:: humchan?noaction -> goto scanneditemstate;
fi;
if
:: humchan?pressstart -> goto disabledHUMActionState;
fi;
if
:: humchan?bagitem -> goto scanneditemstate;
fi;
cschan!allowbag;
if
:: humchan?noaction -> goto scanneditemstate;
fi;
if
:: humchan?bagitem -> goto baggeditemstate;
fi;
baggeditemstate:
startpressed = true;
```

```
itemscanned = true;
itembagged = true;
cschan!noaction;
if
:: humchan?noaction -> goto baggeditemstate;
fi;
if
:: humchan?pressstart -> goto disabledHUMActionState;
fi;
if
:: humchan?bagitem -> goto baggeditemstate;
fi;
cschan!allowpay;
if
:: humchan?noaction -> goto baggeditemstate;
fi;
if
:: humchan?payforitem -> goto paidforitemstate;
fi;
paidforitemstate:
startpressed = true;
itemscanned = true;
itempaid = true;
cschan!noaction;
if
:: humchan?noaction -> goto paidforitemstate;
fi;
if
:: humchan?pressstart -> goto disabledHUMActionState;
fi;
if
:: humchan?leavestore -> goto initialstate;
fi;
if
:: humchan?payforitem -> goto disabledHUMActionState;
fi;
errorstate:
cschan!noaction;
if
:: humchan?noaction -> goto errorstate;
fi;
```

```
if
:: humchan?pressstart -> goto disabledHUMActionState;
fi;
if
:: humchan?scanitem -> goto disabledHUMActionState;
fi;
if
:: humchan?payforitem -> goto disabledHUMActionState;
fi;
if
:: humchan?bagitem -> goto errorstate;
fi;
if
:: humchan?leavestore -> goto errorstate;
fi;
cschan!errorReset;
if
:: humchan?noaction -> goto initialstate;
fi;

}
```

Listing D.1: Sparse CGS for Self-Scan System in Promela

D.2 C_{all} from the Sparse CGS for Self-scan Checkout System

```
begincgs
predicates itembagged;startpressed;itemscanned;itempaid;erred;alarmraised
players HUM;CS
beginallplayeractionlist
beginplayeractionlist HUM
noaction;pressstart;scanitem;payforitem;bagitem;leavestore
endplayeractionlist
beginplayeractionlist CS
noaction;allowpressstart;allowbag;allowpay;allowscan;errorReset
endplayeractionlist
endallplayeractionlist
beginvariablelist
endvariablelist
beginstatelist
beginstate initialstate
beginplayeractionlist HUM
```

```
noaction ; pressstart ; scanitem ; bagitem ; leavestore
endplayeractionlist
beginplayeractionlist CS
noaction ; allowpressstart
endplayeractionlist
beginvalidpropositions

endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
initialstate , noaction , noaction ; initialstate , pressstart , noaction ;
initialstate , noaction , allowpressstart ; pressedstartstate , pressstart , allowpressstart ;
errorstate , scanitem , noaction ; errorstate , scanitem , allowpressstart ; errorstate , bagitem , noaction ;
errorstate , bagitem , allowpressstart ; errorstate , leavestore , noaction ;
errorstate , leavestore , allowpressstart
endtransitions
endstate
beginstate pressedstartstate
beginplayeractionlist HUM
noaction ; scanitem ; payforitem ; bagitem ; leavestore
endplayeractionlist
beginplayeractionlist CS
noaction ; allowscan
endplayeractionlist
beginvalidpropositions
startpressed
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
pressedstartstate , noaction , noaction ; pressedstartstate , scanitem , noaction ;
pressedstartstate , noaction , allowscan ; scanneditemstate , scanitem , allowscan ;
errorstate , payforitem , noaction ; errorstate , payforitem , allowscan ;
errorstate , bagitem , noaction ; errorstate , bagitem , allowscan ;
errorstate , leavestore , noaction ; errorstate , leavestore , allowscan
endtransitions
endstate
beginstate scanneditemstate
beginplayeractionlist HUM
noaction ; bagitem ; scanitem ; payforitem ; leavestore
```

```
endplayeractionlist
beginplayeractionlist CS
noaction ; allowbag
endplayeractionlist
beginvalidpropositions
startpressed ; itemscanned
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
scanneditemstate , noaction , noaction ; scanneditemstate , bagitem , noaction ;
scanneditemstate , noaction , allowbag ; baggeditemstate , bagitem , allowbag ;
errorstate , scanitem , noaction ; errorstate , scanitem , allowbag ;
errorstate , payforitem , noaction ; errorstate , payforitem , allowbag ;
errorstate , leavestore , noaction ; errorstate , leavestore , allowbag
endtransitions
endstate
beginstate baggeditemstate
beginplayeractionlist HUM
noaction ; bagitem ; payforitem ; scanitem ; leavestore
endplayeractionlist
beginplayeractionlist CS
noaction ; allowpay
endplayeractionlist
beginvalidpropositions
startpressed ; itemscanned ; itembagged
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
baggeditemstate , noaction , noaction ; baggeditemstate , bagitem , noaction ;
baggeditemstate , noaction , allowpay ; paidforitemstate , payforitem , allowpay ;
errorstate , scanitem , noaction ; errorstate , scanitem , allowpay ;
errorstate , leavestore , noaction ; errorstate , leavestore , allowpay
endtransitions
endstate
beginstate paidforitemstate
beginplayeractionlist HUM
noaction ; leavestore ; scanitem ; bagitem
endplayeractionlist
beginplayeractionlist CS
```

```
noaction
endplayeractionlist
beginvalidpropositions
startpressed;itemscanned;itempaid
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
paidforitemstate,noaction,noaction;initialstate ,leavestore,noaction;
errorstate,scanitem,noaction;errorstate ,bagitem,noaction
endtransitions
endstate
beginstate errorstate
beginplayeractionlist HUM
noaction;bagitem;leavestore
endplayeractionlist
beginplayeractionlist CS
noaction;errorReset
endplayeractionlist
beginvalidpropositions

endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
errorstate,noaction,noaction;errorstate ,bagitem,noaction;
errorstate,leavestore,noaction;initialstate ,noaction,errorReset
endtransitions
endstate
endstatelist
endcgs
```

Listing D.2: C_{all} for Self-Scan System in CGS specification syntax

D.3 Recommended task specification for using the self-scan checkout system

```
pressstart
scanitem
bagitem
payforitem
```

```
leavestore
```

Listing D.3: Recommended task specification for purchasing a single item

D.4 Recommended task CGS for using the self-scan checkout system

```
begincgs
predicates itembagged;startpressed;itemscanned;itempaid;erred;alarmraised
players HUM;CS
beginallplayeractionlist
beginplayeractionlist HUM
noaction;pressstart;scanitem;payforitem;bagitem;leavestore
endplayeractionlist
beginplayeractionlist CS
noaction;allowpressstart;allowbag;allowpay;allowscan;errorReset
endplayeractionlist
endallplayeractionlist
beginvariablelist
endvariablelist
beginstatelist
beginstate initialstate
beginplayeractionlist HUM
noaction;pressstart
endplayeractionlist
beginplayeractionlist CS
noaction;allowpressstart
endplayeractionlist
beginvalidpropositions

endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
initialstate ,noaction ,noaction;initialstate ,pressstart ,noaction;
initialstate ,noaction ,allowpressstart;pressedstartstate ,pressstart ,allowpressstart
endtransitions
endstate
beginstate pressedstartstate
beginplayeractionlist HUM
noaction;scanitem
```

```
endplayeractionlist
beginplayeractionlist CS
noaction ; allowscan
endplayeractionlist
beginvalidpropositions
startpressed
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
pressedstartstate , noaction , noaction ; pressedstartstate , scanitem , noaction ;
pressedstartstate , noaction , allowscan ; scanneditemstate , scanitem , allowscan
endtransitions
endstate
beginstate  scanneditemstate
beginplayeractionlist HUM
noaction ; bagitem
endplayeractionlist
beginplayeractionlist CS
noaction ; allowbag
endplayeractionlist
beginvalidpropositions
startpressed ; itemscanned
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
scanneditemstate , noaction , noaction ; scanneditemstate , bagitem , noaction ;
scanneditemstate , noaction , allowbag ; baggeditemstate , bagitem , allowbag
endtransitions
endstate
beginstate  baggeditemstate
beginplayeractionlist HUM
noaction ; payforitem
endplayeractionlist
beginplayeractionlist CS
noaction ; allowpay
endplayeractionlist
beginvalidpropositions
startpressed ; itemscanned ; itembagged
endvalidpropositions
```

```
beginvarassigns
endvarassigns
begintransitions
baggeditemstate , noaction , noaction ; baggeditemstate , noaction , allowpay ;
paidforitemstate , payforitem , allowpay
endtransitions
endstate
beginstate  paidforitemstate
beginplayeractionlist HUM
noaction ; leavestore
endplayeractionlist
beginplayeractionlist CS
noaction
endplayeractionlist
beginvalidpropositions
startpressed ; itemscanned ; itempaid
endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
paidforitemstate , noaction , noaction ; initialstate , leavestore , noaction
endtransitions
endstate
beginstate  errorstate
beginplayeractionlist HUM
noaction
endplayeractionlist
beginplayeractionlist CS
noaction ; errorReset
endplayeractionlist
beginvalidpropositions

endvalidpropositions
beginvarassigns
endvarassigns
begintransitions
errorstate , noaction , noaction ; initialstate , noaction , errorReset
endtransitions
endstate
endstatelist
```

```
endcgs
```

Listing D.4: Recommended task CGS

References

[1] http://www.fsel.com/software.html.

[2] http://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[3] http://www.eclipse.org/aspectj.

[4] http://www.java.com.

[5] *Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings*, volume 1248 of *LNCS*. Springer, 1997.

[6] A. Agarwal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. Konig, F. Leymann, R. Muller, G. Pfau, K. Plosser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schimdt, I. Trickovic, A. Yiu, and M. Zeller. *WS-BPEL Extension for People (BPEL4People)*, version 1.0 edition, June 2007.

[7] Thomas Ågotnes, Valentin Goranko, and Wojciech Jamroga. Alternating-time temporal logics with irrevocable strategies. In *TARK*, pages 15–24, 2007.

[8] Rajeev Alur, Pavol Cerný, P. Madhusudan, and Wonhong Nam. Synthesis of interface specifications for java classes. In Jens Palsberg and Martín Abadi, editors, *POPL*, pages 98–109. ACM, 2005.

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

[10] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter Kig, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. *Web Services Business Process Execution Language*. Oasis, version 2.0 edition, April 2007.

[11] Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for petri nets. *Theor. Comput. Sci.*, 3(1):85–104, 1976.

[12] Roy Armoni, Dmitry Korchemny, Andreas Tiemeyer, Moshe Y. Vardi, and Yael Zbar. Deterministic dynamic monitors for linear-time assertions. In Havelund et al. [54], pages 163–177.

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

[14] Pavel Avgustinov, Eric Bodden, Elnar Hajiyev, Laurie J. Hendren, Ondrej Lhoták, Oege de Moor, Neil Ongkingco, Damien Sereni, Ganesh Sittampalam, Julian Tibble, and Mathieu Verbaere. Aspects for trace monitoring. In Havelund et al. [54], pages 20–39.

[15] Pavel Avgustinov, Julian Tibble, and Oege de Moor. On the semantics of matching trace monitoring patterns. In Sokolsky and Tasiran [110], pages 9–21.

[16] Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors. *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418 of *Lecture Notes in Computer Science*. Springer, 2010.

[17] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *VMCAI*, volume 2937 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2004.

[18] Howard Barringer, David E. Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring: From eagleto ruler. In Sokolsky and Tasiran [110], pages 111–125.

[19] Saddek Bensalem and Doron Peled, editors. *Runtime Verification, 9th International Workshop, RV 2009, Grenoble, France, June 26-28, 2009. Selected Papers*, volume 5779 of *Lecture Notes in Computer Science*. Springer, 2009.

[20] C. Bieder, P. Le Bot, E. Desmares, J. L. Bonnet, and F. Cara. Mermos: Edf's new advanced hra method. In *PSAM4*, London, 1998. Springer-Verlag.

[21] Eric Bodden and Patrick Lam. Clara: Partially evaluating runtime monitors at compile time - tutorial supplement. In Barringer et al. [16], pages 74–88.

[22] Eric Bodden, Patrick Lam, and Laurie J. Hendren. Clara: A framework for partially evaluating finite-state runtime monitors ahead of time. In Barringer et al. [16], pages 183–197.

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

[24] Matthew L. Bolton and Ellen J. Bass. Using task analytic models and phenotypes of erroneous human behavior to discover system failures using model checking. In *54th Annual Meeting of the Human Factors and Ergonomics Society*, page 992996, 2010.

[25] R. M. Botting and Chris W. Johnson. A formal and structured approach to the use of task analysis in accident modelling. *Int. J. Hum.-Comput. Stud.*, 49(3):223–244, 1998.

[26] T. Brihaye, A. Da Costa Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts and Bounded Memory. In *LFCS*, volume 5407 of *LNCS*, pages 92–106. Springer, 2009.

[27] Jan Broersen, Andreas Herzig, and Nicolas Troquard. A STIT-Extension of ATL. In *JELIA*, volume 4160 of *LNCS*, pages 69–81. Springer, 2006.

[28] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[29] Mark R. Chassin and Elise C. Becher. The Wrong Patient. *Ann Intern Med*, 136(11):826–833, 2002.

[30] Feng Chen and Grigore Rosu. Java-mop: A monitoring oriented programming environment for java. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 2005.

[31] Feng Chen and Grigore Rosu. Mop: an efficient and generic runtime verification framework. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele Jr., editors, *OOPSLA*, pages 569–588. ACM, 2007.

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

[33] Lori A. Clarke, George S. Avrunin, and Leon J. Osterweil. Using software engineering technology to improve the quality of medical processes. In *ICSE Companion*, pages 889–898, 2008.

[34] Christian Colombo, Gordon J. Pace, and Patrick Abela. Compensation-aware runtime monitoring. In Barringer et al. [16], pages 214–228.

[35] S. E. Cooper, A. M. Ramey-Smith, J. Wreathall, G.W. Parry, D.C. Bley, W.J. Luckas, J. H. Taylor, and M.T. Berriere. A Technique for Human Error Analysis (ATHEANA). Technical Report NUREG/CR-6350, US Neuclear Regulatory Commission, Washington, DC, 1996.

[36] T. L. Crenshaw, A. Tirumala, S. Hoke, and M. Caccamo. A robust implicit access protocol for real-time wireless collaboration. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, 2005.

[37] Paul Curzon and Ann Blandford. From a formal user model to design rules. In Peter Forbrig, Quentin Limbourg, Bodo Urban, and Jean Vanderdonckt, editors, *DSV-IS*, volume 2545 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[38] Paul Curzon and Ann Blandford. Formally justifying user-centred design rules: A case study on post-completion errors. In Eerke A. Boiten, John Derrick, and Graeme Smith, editors, *IFM*, volume 2999 of *Lecture Notes in Computer Science*, pages 461–480. Springer, 2004.

[39] Paul Curzon, Rimvydas Ruksenas, and Ann Blandford. An approach to formal verification of human-computer interaction. *Formal Asp. Comput.*, 19(4):513–550, 2007.

[40] Hasan Davulcu, Michael Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 25–33. ACM Press, 1998.

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

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

[43] Cătălin Dima, Constantin Enea, and Dimitar P. Guelev. Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions. In *GANDALF*, volume 25 of *EPTCS*, pages 103–117, 2010.

[44] Catherine Dufourd, Alain Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, pages 103–115, 1998.

[45] Marlon Dumas and Arthur H. M. ter Hofstede. Uml activity diagrams as a workflow specification language. In *UML*, pages 76–90, 2001.

[46] Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In Behnam Banieqbal, Howard Barringer, and Amir Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer, 1987.

[47] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal basis of fairness. In *POPL*, pages 163–173, 1980.

[48] Hector Garcia-Molina and Kenneth Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, *SIGMOD Conference*, pages 249–259. ACM Press, 1987.

[49] Madhu Gopinathan and Sriram K. Rajamani. Runtime monitoring of object invariants with guarantee. In Leucker [73], pages 158–172.

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

[51] Elsa L. Gunter, Ayesha Yasmeen, Carl A. Gunter, and Anh Nguyen. Specifying and analyzing workflows for automated identification and data capture. In *HICSS*, pages 1–11. IEEE Computer Society, 2009.

[52] Willie Hammer. *Handbook of Product and System Safety*. Prentice Hall Inc., Englewood Cliffs, N.J., 1972.

[53] John C. Harsanyi and Reinhard Selten. A Generalized Nash Solution for Two-Person Bargaining Games with Incomplete Information. In *Management Science*, volume 18 of *Theory Series, Game Theory and Gaming*, pages 80–106, 1972.

[54] Klaus Havelund, Manuel Núñez, Grigore Rosu, and Burkhart Wolff, editors. *Formal Approaches to Software Testing and Runtime Verification, First Combined International Workshops, FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006, Revised Selected Papers*, volume 4262 of *Lecture Notes in Computer Science*. Springer, 2006.

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

[56] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[57] E. Hollnagel. Cognitive Reliability and Error Analysis. In *Elsevier Science Ltd*, Oxford, UK, 1998.

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

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

[60] Yoshinao Isobe1 and Markus Roggenbach. CSP-Prover. http://staff.aist.go.jp/y-isobe/CSP-Prover/CSP-Prover.html.

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

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

[63] C.B. Jones. *Development Methods for Computer Programs Including a notion of Interference*. PhD thesis, Oxford University Computing Laboratory, 1981.

[64] Kari Kähkönen, Jani Lampinen, Keijo Heljanko, and Ilkka Niemelä. The lime interface specification language and runtime monitoring tool. In Bensalem and Peled [19], pages 93–100.

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

[66] Chang Hwan Peter Kim, Eric Bodden, Don S. Batory, and Sarfraz Khurshid. Reducing configurations to monitor in a software product line. In Barringer et al. [16], pages 285–299.

[67] B. Kirwan and L. K. Ainsworth, editors. *A Guide to Task Analysis*. Taylor and Francis, London, 1992.

[68] J. M. Corrigan Kohn, L. T. and M. S. Donaldson. To err is human: Building a safer health system. http://www.nap.edu/catalog/9728.html.

[69] Orna Kupferman and Moshe Y. Vardi. Module checking. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 1996.

[70] Orna Kupferman and Moshe Y. Vardi. Robust satisfaction. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 383–398. Springer, 1999.

[71] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

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

[73] Martin Leucker, editor. *Runtime Verification, 8th International Workshop, RV 2008, Budapest, Hungary, March 30, 2008. Selected Papers*, volume 5289 of *Lecture Notes in Computer Science*. Springer, 2008.

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

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

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

[77] Michael R. Lyu, editor. *Software Fault Tolerance*. John Wiley and Sons Ltd., Chichester, England, 1995.

[78] P. Madhusudan. *Control and Synthesis of Open Reactive Systems*. PhD thesis, University of Madras, Chennai, India, 2001.

[79] Nicolas Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS*, 79:122–128, 2003.

[80] Patrick Meredith, Dongyun Jin, Feng Chen, and Grigore Roşu. Efficient monitoring of parametric context-free patterns. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering(ASE '08)*, pages 148–157. IEEE/ACM, 2008.

[81] Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the MOP runtime verification framework. *International Journal on Software Techniques for Technology Transfer*, 2011. to appear.

[82] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Commun. ACM*, 33(12):32–44, 1990.

[83] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[84] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes. In *Information and Computation*, pages 1–40 and 41–77, September 1992.

[85] C.M. Mitchell. GT-MSOCC: A Domain for Research on Human Computer Interaction and Decision Aiding in Supervisory Control Systems. *IEEE Transactions on Systems, Man and Cybernetics*, 17(4):553–572, July 1987.

[86] F. Mogavero, A. Murano, and M. Y. Vardi. Relentful strategic reasoning in alternating-time temporal logic. In *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 371–386, 2010.

[87] Louise E. Moser, P. M. Melliar-Smith, G. Kutty, and Y. S. Ramakrishna. Completeness and soundness of axiomatizations for temporal logics. without next. *Fundam. Inform.*, 21(4):257–305, 1994.

[88] D. A. Norman. Errors in human performance. Technical Report 8004, University of California, San Diego, Center for Human Infromation Processing Report, 1980.

[89] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, October 2003.

[90] Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, Fakultat fur Mathematik und Physik, Technische Hochschule Darmstadt Germany, 1962.

[91] Lee Pike, Alwyn Goodloe, Robin Morisset, and Sebastian Niller. Copilot: A hard real-time runtime monitor. In Barringer et al. [16], pages 345–359.

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

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

[94] Harald Psaier, Lukasz Juszczyk, Florian Skopik, Daniel Schall, and Schahram Dustdar. Runtime behavior monitoring and self-adaptation in service-oriented systems. In *SASO*, pages 164–173. IEEE Computer Society, 2010.

[95] Frank Puhlmann and Mathias Weske. Using the π-calculus for formalizing workflow patterns. In *Business Process Management*, volume 3649, pages 153–168, 2005.

[96] Jens Rasmussen. Skills, rules, and knowledge: Signals, signs, and symbols and other distinctions in human performance models. In *SMC*, volume 13, pages 257–267, 1983.

[97] J.T. Reason. *Human error*. Cambridge University Press, 1990.

[98] John H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.

[99] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[100] Grigore Rosu, Feng Chen, and Thomas Ball. Synthesizing monitors for safety properties: This time with calls and returns. In Leucker [73], pages 51–68.

[101] Rimvydas Ruksenas, Paul Curzon, Jonathan Back, and Ann Blandford. Formal modelling of cognitive interpretation. In Gavin J. Doherty and Ann Blandford, editors, *DSV-IS*, volume 4323 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

[102] Junzo Sato. Aircraft Accident Investigation Report: Japan Airline Flight 907,Japan Airline Flight 958: A Near Mid-Air Collision over the sea off Yaizu City. *Aircraft and Railway Accident Investigation Commision*, Japan.

[103] Pierre-Yves Schobbens. Alternating-time logic with imperfect recall. *ENTCS*, 85(2):82 – 93, 2004. LCMAS 2003.

[104] A. Shepherd. Analysis and training in information technology tasks. *Task Analysis for Human-Computer Interaction*, pages 15–55, 1989.

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

[106] Jocelyn Simmonds, Marsha Chechik, Shiva Nejati, Elena Litani, and Bill O'Farrell. Property patterns for runtime monitoring of web service conversations. In Leucker [73], pages 137–157.

[107] Rishabh Singh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning component interfaces with may and must abstractions. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 527–542. Springer, 2010.

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

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

[110] Oleg Sokolsky and Serdar Tasiran, editors. *Runtime Verification, 7th International Workshop, RV 2007, Vancover, Canada, March 13, 2007, Revised Selected Papers*, volume 4839 of *Lecture Notes in Computer Science*. Springer, 2007.

[111] Christian Stefansen. Smawl: A small workflow language based on ccs. In *CAiSE Short Paper Proceedings*, 2005.

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

[113] Juoko Suokas. The role of safety analysis in accident prevention. In *Accident Analysis and Prevention*, volume 20(1), pages 67–85, 1988.

[114] A. D. Swain and H. E. Guttman. Handbook of human reliability analysis with emphasis on nuclear power plant applications. Technical report.

[115] Yoichi Takayama, Ernie Ghiglione, Scott J. Wilson, and James Dalziel. Collaborative activity human workflow (cahw) in eresearch.

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

[117] Hiroaki Tomita. Accident investigation into a near mid-air collision. *Investigator General*, ARAIC, Japan, 2005.

[118] W. M. P. van der Aalst. Verification of workflow nets. In *ICATPN* [5], pages 407–426.

[119] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

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

[121] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

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

[123] Markus Voelter. Aspectj-Oriented Programming in Java. *Java Report*, January 2000.

[124] J. C. Williams. A Data-based Method for Assessing and Reducing Human Error to Improve Operational Performance. In *Proceedings of IEEE 4th Conference on Human Factors in Power Plants*, Monterey, CA, June 1988.

[125] Peter Y. H. Wong and Jeremy Gibbons. A process-algebraic approach to workflow specification and refinement. In Markus Lumpe and Wim Vanderperren, editors, *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2007.

[126] Ayesha Yasmeen and Elsa Gunter. Robustness for Protection Envelopes with Respect to Human Task Variation(to appear). In *IEEE International Conference on Systems, Man and Cybernetics (SMC), Alaska, USA*, 2011.

[127] Yang Yu and Xiaohui Li. A workflow model with temporal logic constraints and its automated verification. In *GCC*, pages 681–684. IEEE Computer Society, 2007.

[128] Antonia Zhai, Guojin He, and Mats Per Erik Heimdahl. Hardware supported flexible monitoring: Early results. In Bensalem and Peled [19], pages 168–183.

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

[130] Wenchao Zhou, Oleg Sokolsky, Boon Thau Loo, and Insup Lee. *MaC*: Distributed monitoring and checking. In Bensalem and Peled [19], pages 184–201.