TOWARD AUTOMATIC GENERATION OF USER INTERFACES: ABSTRACTION OF INTERNAL STATES AND TRANSITIONS

Asaf Degani
NASA Ames Research Center
Moffett Field, CA 94035-1000, USA

Michael Heymann
Technion, Israel Institute of Technology
Haifa, Israel 32000

Abstract: In this paper we discuss a formal approach and methodology for reducing and abstracting the internal states and transitions of a (discrete-event) system representation. The resulting abstracted description, called the "user model," forms the foundation of the user interface as it formulates the necessary modes, states and transitions that drive the interface.

Keywords: Keywords: Computer Interfaces, Human Factors, Formal Methods.

1. INTRODUCTION

From a formal perspective, the user interface can be viewed as an abstraction of the machine's behavior. In every user interface, from a portable CD player to a control panel of an aircraft autopilot, the modes, indications, and parameters seen on the screen are always an abstraction of the states of the underlying machine. Consequently, the dynamics observed on the interface (e.g., mode transitions) are also an abstraction of the far more complex internal dynamics that take place inside the machine. As such, a fundamental aspect of interface design involves an intricate process of abstracting what's irrelevant to the user, retaining what is, and insuring proper correspondence between the abstracted interface and the internal working of the machine. The end result of this process is the information content of the interface which then forms the basis for the graphical user interface (See Heymann and Degani 2007 for a review of formal methods approaches and techniques in user-interface design).

From this perspective, the designer's goal is to strike a fine balance between providing insufficient information on the one hand and providing superfluous information (and overloading the user) on the other. When the information on the interface is insufficient, the user may not be able to perform a specified task correctly (e.g., determine the current mode of the machine and anticipate its next mode as a consequence of user interaction). As a result, either the user will be unable to perform the desired task altogether or there will be unexpected, faulty, and potentially dangerous outcomes.

In situations where the interface provides superfluous information, the practical implications are that the interface may require complex interaction sequences that make it difficult and time consuming to use the system or device. Overburdening the user with too much information on the screen also has the penalty of hindering the users' ability to focus on what's important and detect meaningful patterns. Specifically, we strive for interfaces (and user-manuals) that are not only correct, but also succinct. Generally speaking, we would prefer to have a small set of modes rather then a large set of modes while operating a device. Likewise, we would prefer a short and simple sequence of interaction for accomplishing a given task rather than a convoluted sequence that requires much attention with lots of buttons and key presses.

2. FORMAL ASPECTS OF USER INTERACTION

The correspondence between the machine's underlying behavior and the (abstracted) information that is provided to the user can be formally described and analyzed by considering the following three elements: the *machine*, the *user's tasks*, and the *user's model* of the machine (which is the foundation of the user interface).

We consider machines that interact with their human users, the environment, and can act automatically. A widely used formalism to model machines (e.g., computers) is to describe them as state transition systems. In general, we consider two types of transitions: manually triggered (by the user) and automatically triggered (by the machine). With respect to automatic, we have two sub-categories: those that are triggered by the machine's internal dynamics (e.g., timed transitions) and those that are triggered by the external environment (e.g., the way an A/C compressor is activated when the temperature reaches a set value).

Generally speaking, users interact with a machine to achieve a specific set of tasks (Parasuraman *et al.*, 2000). These *user's tasks* may vary widely, and range from programming consumer electronic devices, such as VCRs, to interacting with web-browsers, and all the way to operating automated and safety-critical systems such as medical devices and navigation systems onboard ships and aircraft. With respect to monitoring information and controlling automated systems, typical tasks involve monitoring mode changes, manually interacting with the system, and supervising a system such that it does not enter into an illegal state. All these tasks can be formally described by partitioning the entire machine's state-space into disjoint clusters that we shall call here "specification classes." A specification class is a set of internal states which the design team determined that the user need not distinguish among. The distinction is typically done by task analysis and by obtaining inputs from expert users. Next, the design team specifies the task requirements. For example, one task requirement, which is common to almost all automated systems that are supervised by humans, is for the user to track these specification classes, unambiguously.

Manufacturers normally provide users with information about the working of the machine by means of user-manuals. Here the manufacturers describe the functions of the machine and its behavior as a consequence of user action and environmental conditions. Most verbal statements for consumer electronics as well as more complex systems take the following form: "When the machine is in mode A and button *x* is pushed, the machine transitions to mode B." These series of fragmented statements describe to the user how the machine works, as well as how he or she is expected to interact with it. We refer to this formal description of the interface indications, and of the transitions and events that drive it, as the *user model* of the machine.

1.1 Interface Correctness Criteria

For the purpose of the analysis, the machine model and user's tasks must be fully specified. (Our only assumption is that the machine's behavior is deterministic and the user's tasks are within the machine's abilities). This leaves the user model (and the interface that is embedded in it) as the focus of the analysis.

One immediate observation about the correctness of user interfaces is that the machine's response to user-triggered events must be deterministic. That is, there must not be a situation wherein, starting from the same mode, an identical user event will sometimes transition the system into one mode and at other times into another. Additionally, there are three user-interface correctness criteria that must be satisfied in the process of abstracting the underling machine behavior and generating user models: An interface is correct if there are no error states, no restricting states, and no augmenting states: (1) An *error state* occurs when the user interface indicates that the machine in one mode when, in fact, the machine is in another. Interfaces with error states lead to faulty interaction. Frequently (but not always), error states are caused by the presence of non-deterministic responses to user interaction. (2) A *restricting state* occurs when the user can trigger mode changes that are not present in the user model and interface. Interfaces with restricting states tend to surprise and confuse users. (3) An *augmenting state* occurs when the user is told that certain transitions are available and can be manually triggered, when in fact, they cannot be executed by the machine (or are disabled). Interfaces with augmenting states tend to puzzle users and have contributed to operational errors. All three criteria can be expressed mathematically, and therefore can be dealt with using formal methods of analysis (see Degani and Heymann 2002 for the theoretical foundation of these criteria, methods and tools for using them, and application to the verification of a modern autopilot).

3. ALGORITHMIC APPROACH

The objective of the abstraction methodology is to derive a user model that is correct for the specified tasks; namely, that is deterministic and free of error-, restricting- and augmenting-states. A second requirement is that this user model must be succinct. The methodology for satisfying these two requirements focuses on a systematic method for reducing the machine model into a smaller model and then abstracting some of its transitions and events.

The conceptual approach for generating correct and succinct user-models is based on the fact that not all the system's internal states need to be individually presented to the user. Thus, while the user-model must enable us to operate the machine correctly (i.e., unambiguously track the specification classes visited by the system), the user need not track every internal state of the machine. From an interface design standpoint, two states A and B can be grouped together on the display and represented as a single user-model state if the intrinsic details of whether the current internal state is A or B are inconsequential to the user.

Formally we say that two internal states need not be distinguished, whenever (1) they belong to the same specification class, (2) each user triggered event that is available and active in one of the states is available

and active also in the other, and (3) whenever starting from either of the two states and triggered by the same event sequence, the state pairs visited, respectively, also satisfy conditions (1) and (2). Such state pairs that need not be distinguished by the user are referred to as *compatible*.

An efficient algorithm for computing such compatible and incompatible state-pairs is based on the use of *merger tables* (Kohavi, 1978; Paull, and Unger, 1959). A merger table is a table of cells that lists for each state pair of the machine, the set of all distinct state pairs that are reached through a single common transition event. For a machine with *n* states, there are *n*[n-1]/2* cells in the table, and by iteratively stepping through the table one event transition at a time, we progressively detect all incompatible state pairs, thereby "resolving" the table. Finally, all the state-pairs that are not found to be incompatible are designated as compatible. From these pairs, all the largest possible sets of compatible states—i.e., triplets, quadruples, quintuples, etc., called *maximal compatibles*—are then computed.

The next step consists of constructing a reduced user-model. The user-model's states are comprised of maximal compatible state sets which constitute the user model's building blocks. In general, however, not all the maximal compatibles need to be part of the reduced model; frequently the designer has more than one choice in selecting appropriate compatible sets. The key to a suitable selection is that the selected set must constitute a *cover* of the original machine's state set. That is, each state of the original machine must be a member of at least one selected maximal compatible (this constitutes the cover property). The state set is selected by first choosing maximal compatibles that constitute a minimal cover of the machine's state set (i.e., none of the selected maximal compatibles can be omitted from the selected set without violating the cover property). If necessary, additional maximal compatibles are incrementally added to the selected minimal cover so as to insure that the set of target states of each transition emanating from a maximal compatible is included in some maximal compatible of the selected set. (In the worse case this incremental addition of maximal compatibles will terminate when all maximal compatibles are chosen.

Once the state set of the reduced model has been selected as just described, the next step is to determine the transitions in the reduced model. These are defined so as to be consistent with the original machine model and with the partition of the state set into specification classes. Three subtle issues arise in this connection and are discussed in detail in the following sections: (1) transition non-determinism of automatically triggered events that can be safely eliminated from the reduced model, (2) sets of events can be grouped together, and (3) events that can be deleted since their presence in the reduced model is non-effectual. The resultant model constitutes the user model and the required interface is then extracted from this model.

4. CONSTRUCTION OF THE USER MODEL

The machine model of Fig. 1 has 18 states and 42 transitions (some of which are manually triggered, such as *ua* and *ud*; the rest are automatic). Four separate specification classes were defined for this machine: A, B, C, as highlighted in gray in Fig. 1. The task requirement is that the user must be able to identify the current specification class (mode) of the machine and to anticipate the next mode that the machine will enter as a result of his or her interactions.

We apply the algorithmic procedure for computing compatible sets to the machine model of Fig. 1 and its specification classes (See Heymann and Degani, 2007 for the mathematical details of the algorithmic procedure). The algorithm terminates with eight maximal compatibles. These eight maximal compatibles and their respective internal states are listed in Fig. 2.

One immediate observation is that some of the maximal compatibles overlap on internal states. For example, internal states 21, 22, 31 and 32 appear in all first four maximal compatibles, and internal state 11 appears in maximal compatible 1 and 3. Such overlap of maximal compatibles is quite common, and frequently implies the existence of multiple candidate user models. And indeed, in this example, there are two candidate user models that contain all the internal states of the machine and thereby satisfy the *cover property* discussed earlier: One consists of the maximal compatibles 1,4,5,6,7,8 (highlighted in light gray in Fig. 2) while the other consists of 2,3,5,6,7,8 (dark gray).

The selection among the various covers and candidate user models cannot, generally, be quantified, as it is based on engineering and human-factors considerations. Here various kinds of design decisions can be brought to bear: The number and intuitive nature of the states and transitions in each candidate, the physical interpretation of the reduced model, the cost and technical considerations in sensing a given event, etc. Of course, when no profound reason exists to prefer one cover over

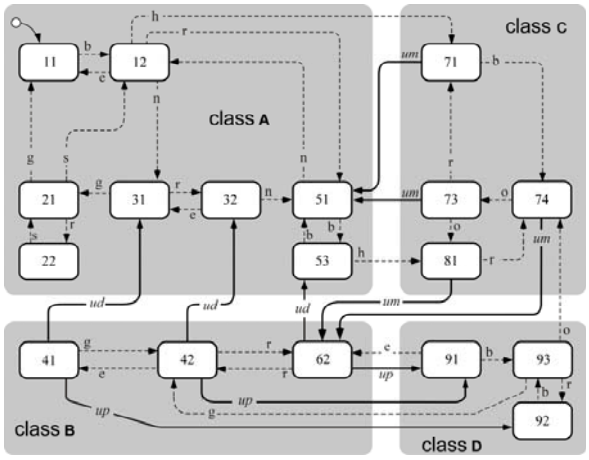

Fig. 1. The machine model.

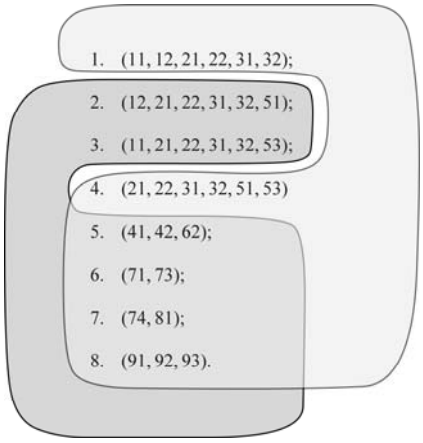

Fig. 2. Eight maximal compatibles and two minimal covers.

another, any cover may be selected. In the following example of how to construct a user model we selected the cover that consists of the maximal compatibles 2,3,5,6,7,8.

We now proceed to construct the user model. We first incorporate the selected maximal compatibles into user model modes depending on their corresponding specification classes. Thus, Maximal compatibles 2 and 3 are designated modes A-1 and A-2 respectively (the index -1 and -2 is arbitrary). Maximal compatible 5 is now mode B. Maximal compatibles 6 and 7 are designated as modes C-1 and C-2 respectively, and the last maximal compatible, 8, is mode D (see Fig. 3 for the mode designations).

Next, we proceed to establish the transitions in the user model. We begin by building a table that lists all user model modes. For each mode, the event labels that emanate form it and all resultant machine model target states are marked (Fig. 3). Once the table in Fig. 3 is established, we can build the transition function for the user model. The following procedure is used: for each mode (e.g., A-1), its constituents event labels (e.g., *h,r,n,e,g,s,b*) are drawn to each mode that includes all the machine model target states (as listed in the parenthesis above each event label in Fig. 3). The resulting user model is depicted in Fig. 4.

	(71)	(51, 22, 32)	(31, 51, 12)	(11, 31)	(11, 21)	(12, 21)	(53)
MODE A-1							
2. (12, 21, 22, 31, 32, 51)	h	r	n	e	g	s	b
	(12, 51)	(11, 21)	(12, 21)	(22, 32)	(31)	(51)	(81)
MODE A-2							
3. (11, 21, 22, 31, 32, 53)	b	g	s	r	e	n	h
	(31, 32, 53)	(42)	(92, 91)	(62, 42)	(41)		
MODE B							
5. (41, 42, 62)	ud	g	up	r	e		
	(74)	(51)	(71)	(81)			
MODE C-1							
6. (71, 73)	b	um	r	o			
	(62)	(73)	(74)				
MODE C-2							
7. (74, 81)	um	o	r				
	(93)	(62)	(74)	(92)	(42)		
MODE D							
8. (91, 92, 93)	b	e	o	r	g		

Fig. 3. User model modes, their respective event labels, and resultant machine model target states.

5. ABSTRACTION OF AUTOMATIC EVENTS

On initial observation of the user model of Fig. 4 we note that there are self loops transitions in every mode. Such self loops are the by product of the reduction process—indicating internal state transitions that take place inside a mode (e.g., events *r,n,s* in mode A-1). But for the purpose of tracking transitions among the modes this information is superfluous. Therefore, the event labels on these self loops can be (judiciously) abstracted from the user model as we shall discuss shortly.

5.1 Non Determinism

Another observation is the presence of non-deterministic transitions between modes. For example, in Fig. 4, the event r emanates from mode A-2 to both A-1 as well as a self loop to A-2. Note, however, that this non-determinism only occurs in modes within the given specification class (A in this case) and therefore acceptable from our perspective because it does not lead to non-deterministic transitions *between* specification classes; and hence cannot lead to an error state condition.

Such non-determinism of automatic events (we will discuss issues concerning manually triggered events later) is an opportunity to further abstract events from the user model. This, again, is because for the purpose of tracking the specification classes, it does not matter if the system remains in A-2 or transition to A-1 (since they both belong to specification class A). We can thus eliminate this non-determinism by deleting one of the *r* events (e.g., either the one to A-1 or the self loop). In general, we would prefer to delete transition between modes and keep transition that self loop around a mode. This is because self-loop transitions do not cause mode transitions – thereby reducing the amount of mode switching that takes place on the interface. (This is an advantage with respect to the reducing the cognitive load on the user). Thus, when the redundant event label *r* from A-2 to A-1 is deleted, *r* remains only in self-loops. Along the same lines, we delete event label *e* from the transition line from A-2 to A-1, leaving it only in the self loop around A-2.

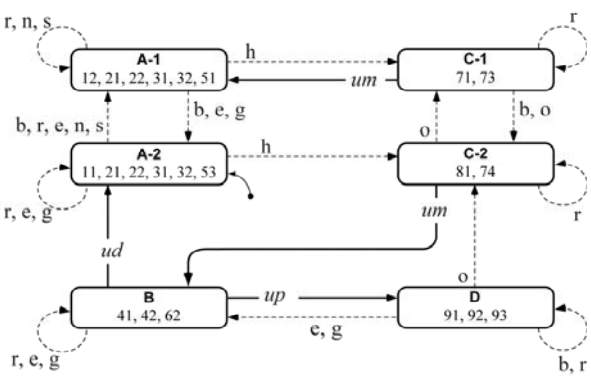

Fig. 4. User model modes, transitions, and all event labels.

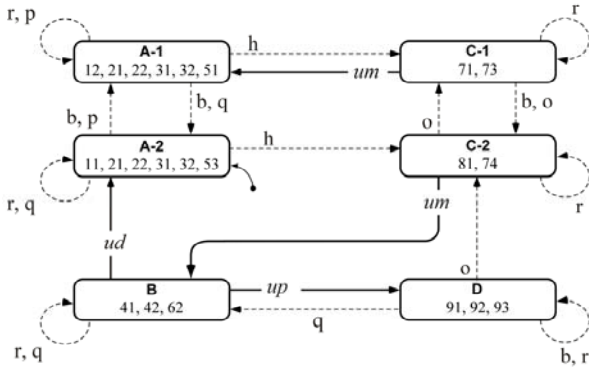

Fig. 5. Abstraction of non-determinism and event groups.

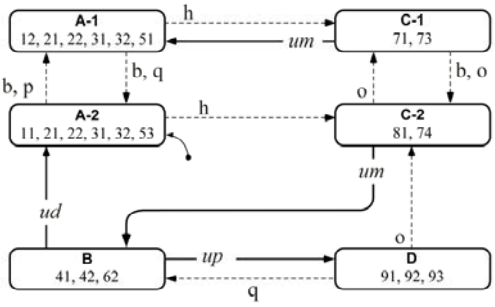

Fig. 6. The final user model. After abstracting for non-deterministic, group, and self-loop events

5.2 Event Groupings

Next, we consider groups of events. We note that in Fig. 4 event labels *e* and *g* always come together. Nowhere in the reduced model do we see *e* or *g* separately; whenever label *e* is enabled in a mode so is *g*. Such groups of events (pairs, triplets, quadruples, and so on) that *always* appear together in transitions can be abstracted into single representative label. In the example, event labels *e* and *g* (outgoing from mode A-1, A-2, B, and D) can be abstracted into an event that we (arbitrarily) label as *q*. Similarly, events *n* and *s* (outgoing from A-1 and A-2) are abstracted into *p*. The outcome of this non-deterministic and event groupings abstraction appears in Fig. 5.

5.3 Non Effectual Events

As a result of deleting event label *r* between mode A-2 and A-1, *r* now appears only in self loops throughout the user model of Fig. 5. Generally speaking, automatic events that occur *only* in self-loops have no effect on the abstracted user model because from the user's point of view, nothing changes in the system's modes as a result of their occurrence. Hence in many cases (but not always, e.g., timing events) we have an opportunity to judiciously remove them from the user model. In practical terms what this means is that event *r* need not be transmitted to the interface at all. Furthermore, if it is difficult or expensive to sense event *r* within the machine, it is possible to avoid sensing it altogether.

In addition to *r*, other automatic event labels appear on self-loop transition in Fig. 5. These are event labels *p* (on A-1), *q* (on A-2 and B), and *b* (on D). But unlike event *r* that can be eliminated completely from the user model because it appears only on self loops—events *p*, *q*, and *b* also appear on other transitions (e.g., between A-1 and A-2). Therefore, events *p*, *q*, and *b* cannot be completely removed (as we did with respect to event *r*). Nevertheless, it is possible to delete them when they occur on self loops. In the model presented in Fig. 6, all these non effectual (self loop) events were removed. This user model, which is both correct and succinct, contains 6 modes, 12 transitions, and 8 different event labels

5.4 Further Elaboration on Self-Loop Events

The issue of deleting self-loop events is rather subtle and requires additional elaboration: Formally speaking, it is indeed possible to eliminate every automatic event on self loops as we did here. Many automotive systems design use this approach when it comes to feedback about the internal state of the vehicle. For example, consider the automatic transmission system of modern cars. While the car is in automatic, drive ("D") mode, internal gear shifts among 1st, 2nd, 3rd, 4th take place. (These internal gear shifts are analogous to the internal state transition within the modes in our example). Nevertheless, these shifts, or transitions, are not annunciated in any way to the driver. Apparently, there is a consensus among car manufacturers that these internal transitions should not be presented.

However, in some cases it may be advantageous to provide information about internal state changes in the user manual and/or perhaps provide some annunciation about their occurrence on the interface. For example, in automated control systems onboard aircraft, there are situations where pilots demand some form of feedback about internal state changes so as to increase their situation awareness. One common solution is to have a certain autopilot mode blink during internal state transitions. In general, it is up to the design team to decide, based on human performance and situation awareness considerations, whether they choose to remove events from self loops or not. The decision cannot be aided by any formal method.

6. ABSTRACTION OF MANUAL EVENTS

The user's task is to track the behavior of the underlying systems (among the various specification classes) as well as to control it manually from one specification class to another. With respect to control, the user would need to know, which user events can be triggered and when, and what will be the ensuing mode. Thus, in mode B the user can trigger either *ud* or *up*, leading the system to either A-2 or to D, respectively. In C-1 and C-2 the user can trigger event *um*, leading the system to modes A-1 or B.

In the reduction process and construction of the user model we treat manually triggered events differently than automatically triggered events. This is due to the *augmenting state* criterion discussed earlier. When a mode abstracts away a set of internal states that have outgoing manual transitions, it is necessary that all the states share the same event label(s), all leading to the same target mode. For example, consider mode B in Fig. 6. Its internal states (41, 42, and 62) all share the same event label *up* leading to mode D (see Fig. 1). Hence no matter which internal state is active, the system always transition to mode B after event *up*. If, for the sake of exposition, state 41 would not have had event *up*, it would have been a violation of the augmenting state criterion to combine all three internal states together into a single mode. This is because a user, who would expect the system to always transition to state D upon manually triggering event *up* while in mode B, would become confused (when the active state is 41).

The above requirement is only for manually triggered events. For automatic transitions, where the user is only monitoring mode changes, the augmenting state criterion is not enforced. Hence, generally speaking, systems that have many automatic transitions are more likely to abstract than systems that have mostly manually triggered transitions.

Finally, recall that the user model of Fig. 6 is based on a cover that contained maximal compatibles 2,3,5,6,7,8 (see Fig. 2). In this case the models constitute a minimal cover. Had we chosen the second candidate (consisting of maximal compatibles 1,4,5,6,7,8) as the basis for our user model, we would have had to increment the cover with maximal compatible 2. This would have to be done because one set of target states only exists in maximal compatible 2. Therefore, the new candidate, (now consisting of maximal compatibles 1,2,4,5,6,7,8) is no longer a minimal cover, but rather an irreducible model.

7. CONCLUSIONS

The approach presented here for abstracting internal states and their transitions is guided by two primary objectives: First and foremost was that the user model must be correct. Second, the selected abstraction should be minimal, or succinct, in terms of the amount of information (e.g., mode annunciations) necessary to control the system. For simplicity and clarity of exposition, we have confined our discussion to machine models that are based on discrete events systems. Nevertheless, the focus of this work is not on a particular modeling formalism and notation. Rather, it is on the ideas that they encapsulate and on the correctness criteria that we need to insure. As such, the approach and methodology can be extended to other discrete event formalisms as well as to hybrid systems models that have both continuous and discrete behaviors (see, for example, the hybrid system modeling and verification approach used in Oishi *et al.* 2003).

In principle, the computation of maximal compatibles for very large systems with thousands of states can become exponentially complex and eventually, computationally intractable. Nevertheless, there are many algorithmic techniques to deal with this problem (e.g., Kam *et al*. 1997). Using a computerized tool the abstraction methodology described in this paper has been successfully applied to machine models with more than 500 internal states. It may be possible, by improving the efficiency of the reduction algorithm, to apply it to larger machines.

We believe that this issue of correct and succinct abstraction is important for both current and future systems. As systems become larger and more complex, there is a growing need for rigorous and systematic methods for user interface design. Specifically, formal and heuristically-based methods for extraction, abstraction, integration, and organization of information for the purpose of display need to be developed to support future adaptive interfaces (that reconfigure themselves based on the situation, the system's state, and the user's information requirements). We believe that the formal methods community can contribute to the development of such methods and tools.

REFERENCES

Degani, A. and Heymann, M. (2002). Formal Verification of Human-Automation Interaction. *Human Factors*, **44**(2), 28-43.

Heymann, M., and Degani, A. (2007). Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm. *Human Factors*, **49**(2), 311–330.

Kam, T., T. Villa, R. Brayton, and Sangiovanni-Vincentelli, A. (1997). Implicit Computation of Compatible Sets for State Minimization of ISFSM's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **16**(6), 657-676.

Kohavi, Z. (1978). *Switching and Finite Automata Theory*. McGraw-Hill: New York.

Oishi M., Tomlin, C. and Degani, A. (2003). *Discrete abstraction of hybrid systems: Verification of safety and application to user-interfaces*. NASA Technical Memorandum #212803, NASA Ames Research Center: Moffett Field, CA.

Parasuraman, R., Sheridan, T., and Wickens, C. (2000). A model for the types and levels of human interaction with automation. *IEEE Transaction on Systems, Man, and Cybernetics - Part A: Systems and Humans*, **30**(3), 286-297.

Paull, M. and Unger, S. (1959). Minimizing the number of states in incompletely specified sequential switching functions. *Institute of Radio Engineers Transactions on Electronic Computers*, 356-367.