Verifying High-Confidence Interactive Systems: Electronic Voting and Beyond

Sanjit A. Seshia

EECS Department, UC Berkeley
sseshia@eecs.berkeley.edu

Abstract. Human interaction is central to many computing systems that require a high level of assurance. We term such systems as *high-confidence interactive systems*. Examples of such systems include aircraft control systems (interacting with a pilot), automobiles with self-driving features (interacting with a driver), medical devices (interacting with a doctor), and electronic voting machines (interacting with a voter). A major challenge to verifying the correct operation of such systems is that it is difficult to formally specify the human user's view of correct operation and perception of the input/output interface. In this paper, we describe a promising approach towards addressing this challenge that combines formal verification with systematic testing by humans. We describe an illustrative application of this approach to electronic voting in the U.S., and outline directions for future work.

1 Introduction

High-confidence computer systems are those that require a high level of assurance of correct operation. Many of these systems are *interactive* — they interact with a human being — and the human operator's role is central to the operation of the system. Examples of such systems include fly-by-wire aircraft control systems (interacting with a pilot), automobiles with driver assistance systems (interacting with a driver), medical devices (interacting with a doctor, nurse, or patient), and electronic voting machines (interacting with a voter). The costs of incorrect operation in all such systems can be very severe. Given the central role of the human operator/user in these systems, correct operation necessarily involves the human-computer interface; in fact, problems in this interface are often the source of failures. For instance, the U.S. Federal Aviation Administration has attributed several incidents, including fatal crashes, to problems in the human-computer interface [3]. Similarly, human errors in medical device use account for a large portion of medical errors, and many of these errors are due to poor design of the interface [5, 6]. It is therefore essential to develop techniques to ensure correct operation of such high-confidence interactive systems.

Formal methods appear to provide the perfect fit for this need. Techniques such as model checking and automatic theorem proving have made tremendous advances over the past several years, with successful applications to the verification of hardware, software, and even biological systems. However, they are only applicable to systems where all the parts are formally specifiable. This presents a problem for interactive systems, where it is difficult or even impossible to formally specify the human user's view of correct operation and perception of the input/output interface. For instance, given a bitmap image on a touch-sensitive screen, automatically recognizing which portions of the screen a human would expect to form a touchable region (e.g. button) might require non-trivial image processing. Testing by humans is well-suited to checking that the system performs according to their expectation, since it eliminates the need to model the human operator. However, a limitation of conventional testing is that it is not exhaustive and it therefore can only find bugs; it cannot guarantee their absence.

In this paper, we advocate for a new approach that enables *the principled design of high-confidence interactive systems where correctness is certified through a combination of formal verification and testing by humans*. The approach has three main steps. First, we need to identify design principles that ease the task of verification and testing. Second, given a set of verification tasks, we need scalable algorithmic methods to tackle them. Third, a rigorous testing protocol must be specified for humans that also uses a tractable number of tests (ideally, low-degree polynomial in the size of the design, since running each test involves possibly several hours of human effort). We have performed an initial application of the approach to direct-recording electronic voting for U.S. elections [7].

Two key elements of the approach are *integrating design and verification* and using *formal verification to reduce the testing burden*. We propose that systems must be designed in a component-based manner following the principles of *determinism*, *independence*, and *unambiguity of the I/O interface*. Unambiguity of the output, for example, means that the system output is a 1-1 function of a subset of "core" state variables defined by the designer (possibly along with the system input), and nothing else. Determinism ensures that these core state variables evolve as a function only of their previous values and the system input. For modularity, the core state variables are partitioned amongst different system components. Independence ensures that updating the state of one component does not change the state of other components.

The above design principles are not new in and of themselves. However, the way in which we *combine* them for design and *verify* them on an implementation is novel. We verify *determinism*, *independence*, and *unambiguity* on the system code using encodings to satisfiability modulo theories (SMT) formulas, which are then proved automatically [2]. The results of the above verification is used to reduce the amount of human-driven testing to a tractable number. Our approach uses rigorous test coverage criteria to derive "test scripts" which humans can then use to evaluate the correctness of the system through testing.

In the rest of this paper, we sketch out the approach using electronic voting as an illustrative application domain (Section 2), and outline directions for future work (Section 3).

2 Electronic Voting

We begin by describing one of our key motivating applications: electronic voting. The work described in this section is joint with several colleagues and has been published earlier [7].

2.1 Preliminaries

Electronic voting is increasingly coming into use in the U.S. and around the world. It has brought with it concerns about reliability, accuracy, and trustworthiness. Existing electronic voting systems can be complex systems, often consisting of hundreds of thousands of lines of code, and a single bug anywhere in the code could potentially cause votes to be lost, misrecorded, or altered. As a result, it is difficult for independent evaluators to be confident that these systems will record and count the votes accurately. Moreover, in order to completely verify the voting machine, it is necessary to also verify the interface to human voters, i.e., that the operation of the voting machine is consistent with the behavior expected by voters.

The kind of voting machine that we focus on here is known as a direct-recording electronic (DRE) voting machine (although the principles we use here are applicable elsewhere). A DRE voting machine is one where voters interact with the machine to make their selections and then the votes are recorded electronically. The most familiar example is a touchscreen voting machine, where the voter interacts with a graphical user interface displayed on the screen by software running on the voting machine. The voter presses at various locations on the screen to register her selections, and the voting software records the voter's selections once she is ready to cast her ballot. DREs are widely deployed throughout the US: for instance, in 2008 and 2010 DREs were used by approximately 33% of registered voters [1, 9]. While DRE's are commonly thought to be large, complex machines, in preliminary work [7] we have shown that a functional DRE can be designed as a finite-state machine directly in hardware, in custom Verilog code, so that there is no operating system or runtime software environment to verify.

Before we get into the notion of correctness for a voting machine, here are some voting-related terms that are used throughout the discussion.

Contest: A single race, such as for President, that a voter will vote on.

Ballot: The physical or electronic representation of all contests that a voter will be deciding on election day.

Candidate: A choice in a particular contest. The voter will typically make one or more selections from among two or more candidates for each contest on the ballot.

Voting Session: A voter's interaction with the machine from the time they are given a new ballot until the time their entire ballot is stored in non-volatile memory, i.e., until the time they cast the ballot. A session in U.S. elections typically comprises voting on several contests.

Cast: Casting a vote refers to the action taken at the end of a voting session that causes the selections made in all contests to be irrevocably written to non-volatile memory. Making a selection in a particular contest and moving on to the next contest is *not* considered casting a vote.

Selection State: The state representing the set of all candidates currently selected in a particular contest.

Button: A (usually rectangular) region on the screen. Touching anywhere within this region activates a particular functionality of the machine. The corresponding part of the screen image is often designed to provide the appearance of a physical button.

Given the above terms, consider the notion of correctness for a single voting session: given a series of inputs (button presses) from the human voter, the machine must record votes in accordance with the *expectation of the human voter*.

A human voter's expectation is difficult to *completely* specify formally. However, it is possible to *specify at least part of it formally*, as a state machine describing how the voting machine must update its internal state: for instance, how the set of candidates currently selected should be updated when the voter presses a button, or how the current contest is updated when the voter presses a "next" or "previous" buttong to navigate between contests, or that the ballot is irrevocably cast when the voter presses the "cast" button. This state machine serves to formalize our assumptions about the "mental model" of the user.

However, the specification machine does not specify all human expectations – for instance, what kinds of screen images should be produced by the voting machine. For example, if there is a rectangular region on the screen that displays "Thomas Jefferson" in some readable font, a human might expect that pressing that portion of the screen would select Jefferson, causing Jefferson's name to be highlighted and eventually causing a vote to be recorded for Jefferson if no other selection is subsequently made in this contest. However, because it involves semantic interpretation of the contents of a particular screen image by a human it is not clear how to specify this expected behavior in a precise, mathematical fashion. For instance, given a bitmap image, mechanically recognizing which portions of the screen a human would expect to correspond to a touchable region might require non-trivial image processing; moreover, mechanically determining that the touchable region should be associated with Thomas Jefferson might require computer vision algorithms and other complex computation. Formalizing these kinds of human expectations in a formal logic could be horribly messy, and probably error-prone as well.

For this reason, one might consider using a representative panel of human "test voters" in the validation process. In particular, we ask human voters to cast test votes on the voting machine during pre-election testing. We ask them to check that the machine seems to be working correctly and recording their votes accurately. We assume that if the machine behaves in a way inconsistent with their expectations, they will notice and complain. Consequently, if the voting machine passes all of these tests, then at least we know that the voting machine has behaved in a way consistent with human expectations during those tests.

We propose that *such human-driven testing can be supported by formal verification*. For instance, we can formally verify that the voting machine (as implemented in code or in hardware) behaves *deterministically*. This ensures that the voting machine will behave the same way on election day as it did in pre-election testing.

However, this verification alone is not enough to provide useful guarantees in practice, because the number of tests needed to exhaustively exercise all possible machine behaviors is astronomically large. For instance, in an election with N contests and k choices in each contest, the number of different ways to vote (assuming voters are only allowed to vote for a single candidate in each contest) is k^N, an exponential function of N. Taking into account the possibility to change one's selections in a contest as many times as one likes, the number of ways to interact with the voting machine becomes infinitely large. Clearly, we cannot exhaustively try all of these possibilities in pre-election testing: we need something more selective.

The burden of testing can be reduced by a combination of principled design and formal verification. In the above scenario, we can ensure that only $O(kN)$ tests are needed. Roughly speaking, if the state and behavior for each contest is *independent* of the state of all other contests, it suffices to choose a test suite that attains 100% transition coverage in each individual contest and of navigation between contests, rather than 100% coverage of the whole voting machine's statespace. This can be achieved with $O(k)$ tests per contest, since the state space in a single contest is only of size $O(k)$ (whereas the statespace for the entire voting machine has size $O(k^N)$ and thus would require exponentially many tests to fully cover). Such independence properties can be verified using formal verification.

Each contest can be viewed as a separate *logical component* of the voting machine. Ideally, the design should be structured into logical components in a manner so as to ease the formal verification of *independence* of one logical component on another.

Finally, we also need to verify that the input/output interface of the voting machine is not ambiguous; e.g., that the machine cannot output the same output screen for two different internal states (selection state and contest number). One way of formalizing this is to show that the output bitmap generated by the voting machine code is an injective (1-1) function of the selection state and contest number. Such a *injectivity* property can also be verified by formal verification.

To summarize, verifying that a voting machine meets human expectations must involve the following steps:

- Formalizing part of the human mental model as a finite-state machine;
- Designing the voting machine using *logical components* so that it satisfies the properties of *determinism*, *independence*, and *unambiguity* of input/output;
- Formally verifying that the design actually satisfies the above properties, and
- Testing of the input/output interface by humans, where ideally each logical component (contest) can be tested independently so that the overall number of tests grows polynomially with the number of such components.

For the first step, it is relatively easy to formalize correct operation of the voting machine (informally described above) as a finite-state machine $\mathcal{P}$. We term this state machine the *specification voting machine* — it is intended to capture the typical mental model that a voter has for U.S. elections. Details of this model may be found in our paper [7]; in essence, it specifies the initial state (start in the first contest with no selections), how one can navigate between contests and select (and deselect) candidates within a contest, and what happens when the "cast" button is pressed to finalize one's votes. The important point for the rest of the paper is that the operation can be formalized as a (finite) state machine.

2.2 Verifying Independence, Determinism, and Injectivity

We now describe how SMT solving can be used for verification of the three key properties: independence, determinism, and injectivity. In order to perform formal verification, the implementation (code) of the voting machine is automatically transformed into a finite-state transducer model. For our Verilog implementation [7], this is a trivial step.

Independence and determinism both involve checking that a variable v depends only on some specified set $W = \{w_1, \ldots, w_n\}$ of variables, and *nothing else*. In other words, we must verify that v can be expressed as a deterministic function of the other variables: $v = f(w_1, \ldots, w_n)$, for some function f, or in shorthand, $v = f(W)$. Put another way, we want to check that v deterministically depends on W, and only W, i.e., for every other variable $x \notin W$, v is conditionally independent of x given W. We verify this kind of property by formulating it as a Boolean satisfiability (SAT) problem (for completely bit-level designs) or as a satisfiability modulo theories (SMT) problem for designs specified at higher levels of abstraction.

The first step is to encode a step of the transducer as a Boolean formula. We begin by introducing some notation. We assume there is a set S of state variables, so that each valuation of values to these variables corresponds to a state of the system. Similarly, let I be a set of input variables, and O a set of output variables. For each state variable s, let the variable s' denote the previous value of s; let S' denote the set of these variables. Then we can write the transition relation as a function δ, which expresses the state as a function of the previous state and the input via the relation $S = \delta(S', I)$. (This is shorthand for $s_i = \delta_i(s'_1, \ldots, s'_k, i_1, \ldots, i_\ell)$ for $i = 1, \ldots, k$, assuming $S = \{s_1, \ldots, s_k\}$ and $I = \{i_1, \ldots, i_\ell\}$.) Similarly, we assume the output function is modeled as a function ρ, via the relation $O = \rho(S)$. Thus, we can model a step of the transducer from state S' to S by the formula

$$\phi(S, S', I, O) \equiv \quad S = \delta(S', I) \wedge O = \rho(S).$$

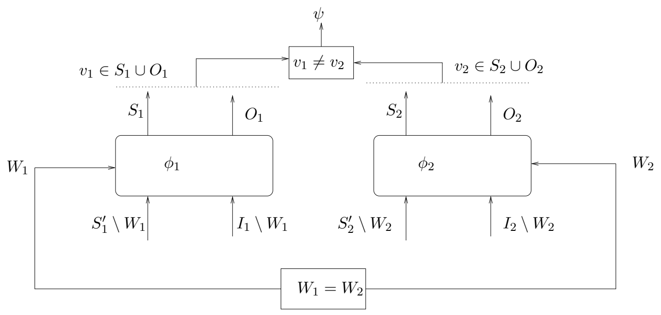

Fig. 1. Satisfiability problem for checking that v deterministically depends on W and nothing else

Now suppose we wish to check that state or output variable v is a deterministic function of a set W of state or input variables. Let S_1, S_2 be two copies of the state variables, I_1, I_2 be two copies of I, and O_1, O_2 be two copies of O. Consider the formula

$$\begin{aligned} &\psi(S_1, S'_1, I_1, O_1, S_2, S'_2, I_2, O_2) \equiv \\ &\quad \phi(S_1, S'_1, I_1, O_1) \wedge \phi(S_2, S'_2, I_2, O_2) \wedge \\ &\quad v_1 \neq v_2 \wedge \forall w \in W \, . \, w_1 = w_2. \end{aligned}$$

Effectively, we make two copies of our model of the system. We then check whether it is possible for v to take on two different values in the two copies, while all variables in W take on the same value in both copies; the answer reveals whether v depends deterministically upon W. In particular, v can be expressed as a deterministic function of W ($v = f(W)$) if and only if ψ is unsatisfiable. Figure 2.2 illustrates this idea. This approach to checking dependence is similar to the technique of using self-composition for checking information flow [8]. The key idea is to formulate non-interference as a 2-safety property.

The property of injectivity can be easily formalized in logic. In general, the outputs O of the transducer is computed as a function $\rho(S)$, where, as above, S is the state of the system. However, for injectivity, we wish to show that ρ is a 1-1 function of a subset of "relevant state variables". We do this in two steps. First, for a candidate set of state variables W, we check (as shown above) that ρ is a function only of variables in W. Then to prove that ρ is an injective function, we additionally need to prove that the following formula is valid:

$$\rho(W_1) = \rho(W_2) \implies (W_1 = W_2)$$

In other words, if two output screens are identical, the relevant state of the system is the same in the two cases.

All of these checks have been performed for the voting machine we designed [7] using the Beaver SMT solver for finite-precision bit-vector arithmetic [4].

2.3 Systematic Human-Driven Testing

A key component of the approach is a systematic protocol for testing the interactive system by human "test users". The main steps are:

1. Prove determinism, independence, and unambiguity (injectivity) properties on the *implementation*, as described above;
2. Define coverage criteria that a test suite must satisfy, and
3. Prove that the two items above taken together ensure correctness of the interactive system.

We now briefly sketch the above approach using our voting machine case study.

A *test input* (or just *test*) is a sequence of button presses involving navigating between contests or selecting candidates that ends in the `cast` button being pressed. Let $\tau_{\mathcal{A}}$ denote the input-output trace exhibited by the implementation machine $\mathcal{A}$ on test input T, and let $\tau_{\mathcal{P}}$ be the trace exhibited by $\mathcal{P}$ on T. We ensure by design and formal verification that $\mathcal{A}$ and $\mathcal{P}$ are deterministic, meaning that for any T, there exists exactly one $\tau_{\mathcal{A}}$ and exactly one $\tau_{\mathcal{P}}$. Denote by I an *input/output interpretation function* that formalizes (i) how a human voter might map regions on the screen to input buttons, and (ii) how the human voter might map the bitmap of an output screen to their perception of the relevant state of the machine (i.e., the current contest and selection state). If $I(\tau_{\mathcal{A}}) = \tau_{\mathcal{P}}$, we say that $\mathcal{A}$ is *correct on test* T or that test T *passes*.

Intuitively, at each step, the tester will check the output screen to make sure that the voting machine appears to have responded correctly, according to their expectations about correct behavior (e.g., after selecting a candidate, the candidate should be highlighted or otherwise appear to be selected). After casting their ballot, the tester will inspect the cast vote record produced by the voting machine (e.g., on a paper readout) and check that it appears to be correct (i.e., it is consistent with the selections the tester has made during this test, according to their interpretation of the test inputs). If any of these checks fail, the human tester will judge the machine $\mathcal{A}$ to have failed; otherwise, the human tester will pass the machine.

A *test suite* $\mathcal{T}$ is a set of complete tests. We say that $\mathcal{T}$ passes if every $T \in \mathcal{T}$ passes.

We assume that if any test fails, the voting system will not be used in an election. Therefore, we wish to identify a condition on $\mathcal{T}$ so that if every test in $\mathcal{T}$ passes, then we can be assured that $\mathcal{A}$ is trace-equivalent to $\mathcal{P}$ after application of the input-output interpretation function. We identify such a sufficient condition on $\mathcal{T}$ below. The condition relies upon the following formally verified properties:

P0: The output function of the voting machine is a injective function of the contest number and selection state of the current contest.
P1: The voting machine is a deterministic transducer.
P2: The state of a contest is updated independently of the state of other contests.
P3: If a navigation button is pressed, the selection state remains unchanged.
P4: If a selection button is pressed, the current contest number stays unchanged.

In addition, we require another property of $\mathcal{A}$ (to be formally verified on the implementation):

P5: The electronic cast vote record that is produced when we cast the ballot is an accurate copy of the selection state for each contest.

All of these properties have been formally verified on the implementation used in our paper [7].

Coverage Criteria We say that a test suite $\mathcal{T}$ satisfies our coverage criteria if the resulting set of traces of $\mathcal{P}$ satisfies the following conditions:

C0: (*Initial State Coverage*) There is a test in which, from the initial output screen z_0, $\mathcal{P}$ receives the `cast` input.
C1: (*Transition Coverage*)
- **(a)** (*Selection transitions*) For every contest i, every selection state s_i within contest i, and every input button b corresponding to a selection, there is some trace where $\mathcal{P}$ receives b in a state (i, s) where the ith component of s is s_i.
- **(b)** (*Navigation transitions*) For every contest i, and every input button corresponding to navigation between contests, there is some trace where $\mathcal{P}$ receives b in a state of the form (i, s).

C2: (*Output Screen Coverage*) For every contest i and every selection state s_i of $\mathcal{P}$ within contest i, there is some trace of $\mathcal{P}$ where *the last transition* within contest i ended at s_i and then at some point thereafter $\mathcal{P}$ receives the `cast` input.

The main correctness theorem we obtain in our paper [7] for the voting machine described therein is that the tests pass iff the machine is trace-equivalent w.r.t. the mental model $\mathcal{P}$:

Theorem 1. *Consider a test suite $\mathcal{T}$ that satisfies coverage criteria C0–C2. Then, $\mathcal{T}$ passes if and only if $\mathcal{A}$ is correct (i.e.,* $\mathrm{Tr}(\mathcal{P}) = \{I(\tau) : \tau \in \mathrm{Tr}(\mathcal{A})\}$*).*

2.4 Extensions

The basic approach outlined in the preceding sections is well-suited for finite-state interactive systems, such as the voting machine. However, even in the domain of electronic voting machines, there is more to be done by including advanced features of voting, such as a summary screen that lists selections made in multiple contests, straight-party voting, where one can cast a vote for all candidates of the same party, instant runoff voting, where one can rank candidates rather than select them, etc. For some of these, we have already developed some initial ideas that can be used to extend the basic approach.

3 Conclusions and Future Work

Even as computing systems are increasingly integrated into our everyday lives, human interaction and operation remains central to their working. In this paper, we describe how a combination of formal verification and systematic testing by humans can help in improving the assurance of these systems. As an illustrative example, we described our work on verification of an electronic voting machine for U.S. elections [7].

A particularly compelling next step is to consider interactive *cyber-physical systems* — systems that tightly integrate computational processes with the physical world, possibly involving networking — that also have humans playing central roles in their operation. Modern automotive, avionics and medical systems are good examples. The technical challenge in these systems vis-a-vis electronic voting arises from the combination and close interaction of continuous and discrete state and dynamics. While the essence of the approach described in this paper, including the properties of independence, determinism, and umabiguity, should remain relevant, extensions are required to deal with the complexity in the state space. Our ongoing work is developing new techniques for such systems.

Another direction for future work involves systems that have multiple humans involved — i.e., teams of human operators or even multiple competing human agents interacting with computing systems, possibly over a network. Altogether, many more advances are needed before we can achieve the goal of high-assurance distributed cyber-physical systems with multiple humans in the loop.

Acknowledgements

This paper describes work conducted jointly with Susmit Jha, Cynthia Sturton, and David Wagner. The author gratefully acknowledges the support of an Alfred P. Sloan Research Fellowship and the National Science Foundation through grants CNS-0644436 and CCF-1116993.

References

1. K. Alexander and P. Smith. Verifying the vote in 2008 presidential election battleground states, Nov. 2008. `http://www.calvoter.org/issues/votingtech/pub/pres2008_ev.html`.
2. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 4, chapter 8. IOS Press, 2009.
3. Federal Aviation Administration (FAA). The interfaces between flight crews and modern flight systems. `http://www.faa.gov/avr/afs/interfac.pdf`, 1995.
4. S. Jha, R. Limaye, and S. A. Seshia. Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In *Proc. Computer-Aided Verification (CAV)*, LNCS 5643. Springer, 2009.
5. L. T. Kohn and J. M. Corrigan and M. S. Donaldson, editors. To err is human: Building a safer health system. Technical report, A report of the Committee on Quality of Health Care in America, Institute of Medicine, Washington, DC, 2000. National Academy Press.
6. J. H. Obradovich and D. D. Woods. Users as designers: How people cope with poor HCI design in computer-based medical devices. *Human Factors*, 38(4):574–592, 1996.
7. C. Sturton, S. Jha, S. A. Seshia, and D. Wagner. On voting machine design for verification and testability. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Nov. 2009.
8. T. Terauchi and A. Aiken. Secure information flow as a safety problem. Technical Report UCB/CSD-05-1396, EECS Department, University of California, Berkeley, Jun 2005.
9. Verified Voting Foundation. America's voting systems in 2010. `http://verifiedvoting.org`.