Runtime Verification of Stochastic, Faulty Systems

Cristina M. Wilcox and Brian C. Williams

Massachusetts Institute of Technology. Cambridge, MA, 02141. USA
cwilcox@alum.mit.edu, williams@csail.mit.edu
http://groups.csail.mit.edu/mers/

Abstract. We desire a capability for the lifelong verification of complex embedded systems that degrade over time, such as a semi-autonomous car. The field of runtime verification has developed many tools for monitoring the safety of software systems in real time. However, these tools do not allow for uncertainty in the system's state or failure, both of which are essential for monitoring hardware as it degrades. This work augments runtime verification with techniques from model-based estimation in order to provide a capability for monitoring the safety criteria of mixed hardware/software systems that is robust to uncertainty and hardware failure.

We begin by framing the problem as runtime verification of stochastic, faulty, hidden-state systems. We solve this problem by performing belief state estimation over the combined state of the Büchi automata representing the safety requirements and the probabilistic hierarchical constraint automata representing the embedded system. This method provides a clean framing of safety monitoring of mixed stochastic systems as an instance of Bayesian filtering.[1]

Keywords: stochastic systems, hidden state, belief state update

1 Introduction

1.1 Runtime Verification for Faulty Embedded Systems

The field of runtime verification seeks to check software correctness at runtime. Runtime verification complements testing methods by providing a framework for automated testing that can be extended into a capability for monitoring a system post-deployment. With a runtime verification capability in place, an operational system can detect deviations from formally specified behavior and potentially take corrective action, providing a capability for fault-tolorance which is desirable for safety critical systems.

[1] This research was supported in part by the Ford-MIT Alliance agreement of 2007, and by a grant from the Office of Naval Research through Johns Hopkins University, contract number 960101.

Runtime verification has also been used in complex *mixed systems*, that is, systems that involve a mix of hardware and software [2, 10]. However, runtime verification for such systems assumes observability of properties to be monitored. We argue that for complex hardware systems, such as a space probe or a car, the system's state is generally unobservable, due to the high cost of sensing all variables reliably. Hence, in order to perform safety monitoring of these mixed systems, this thesis extends proven runtime verification techniques so that they handle systems with hidden states.

To deal with hidden states, we draw upon inference techniques from the field of Model-based diagnosis (MBD) [14], which are based on a model of the system components and constraints. MBD applies conflict-directed search techniques in order to quickly enumerate system configurations, such as failure modes, that are consistent with the model and observations. These techniques are suitable for mixed systems and scale well to systems with large numbers of components [8,14].

A second issue, not directly addressed by runtime verification, is that complex systems with long life cycles experience performance degradation due to seemingly random hardware failure. Many systems function well when manufactured, but may become unsafe over time, especially when they are in use for longer than their intended life span. For example, car owners occasionally fail to have their vehicles inspected promptly, which can result in a component, such as the braking system, receiving more use than it was designed for. We want to be able to detect any breaches of safety due to wear and tear in such a situation.

Thus, this work advocates the use of a plant model that incorporates stochastic behavior [14], allowing wear and tear to be modeled as stochastic hardware failure. With such a model, specification violations resulting from performance degradation can be detected online and recovery action can be taken, such as the removal of unsafe functions.

1.2 Architecture of the proposed solution

We propose a capability for the monitoring of formal specifications for mixed systems that are written in Linear Temporal Logic (LTL) [11]. Linear Temporal Logic is a well studied logic that is similar to plain English and expressive enough to capture many important high-level safety requirements. Additionally, we allow requirements to be written over hidden system states.

Our safety monitoring capability will also have a model of the stochastic, faulty plant captured as a Probabilistic Hierarchical Constraint Automaton (PHCA) [14]. This automaton representation allows for the abstract specification of embedded software, as well as the specification of discrete hardware modes, including known failure modes. Additionally, stochastic transitions may be specified in order to model random hardware failure. Such a model of the system allows the safety monitoring capability to identify hidden system state, including in the case of sensor failure, unmodeled failures, intermittent failures, or multiple faults.

Given sensory information, the safety monitoring capability will then compute online the likelihood that the LTL safety requirements are being met. We

accomplish this by framing the problem as an instance of belief state update over the combined state of the Büchi Automaton and Probabilistic Hierarchical Constraint Automaton, as described in Section 4.2.

Together, LTL and PHCA offer an orderly specification method for performing safety monitoring of mixed stochastic systems. Viewing safety monitoring as belief state update on a hybrid of BA and PHCA state provides a clean framing of the problem as an instance of Bayesian filtering.

1.3 Related Work

Some examples of the successful application of runtime verification techniques in software systems are JPaX, by Havelund and Roşu [6], DBRover, by Drusinsky [3], and MaC [7], by Kim *et al.* In this paper we build on such work by extending these techniques to deal with mixed stochastic systems.

Peters and Parnas [10], and Black [2] have developed monitors for runtime verification of systems that include hardware, but these works do not consider hidden state, which is the primary focus of this paper. Sistla and Srinivas [13], and Sammapun *et al.* [12] present sound monitoring algorithms for software systems exhibiting probabilistic behavior, but neither work is concerned with properties written over hidden system states, and thus their methods do not suffice for the purpose of safety monitoring of mixed systems.

Runtime verification has been moving towards the monitoring of general properties for mixed stochastic systems, but no work we are aware of has attempted to monitor properties written over unobservable system states. Additionally, no work has employed a system model appropriate for faulty hardware systems. The approach presented in this paper provides these novel capabilities.

2 Temporal Logic and Büchi Automata

2.1 Linear Temporal Logic

In this paper we consider safety requirements written in next-free Linear Temporal Logic (LTL) [5,11]. An LTL statement α may be comprised of propositions connected with the usual boolean operators ($\neg$, $\wedge$, $\vee$, $\rightarrow$), as well as the temporal operators always ($\Box$), eventually ($\Diamond$), until ($\mathcal{U}$), and release ($\mathcal{R}$). These operators are formally defined as is usual in the literature.

2.2 LTL to NBA conversion

In order to automate the monitoring of a Linear Temporal Logic statement $\mathbf{\Lambda}$, it may be converted into a nondeterministic Büchi Automaton (NBA) and executed on the finite program trace W. To perform this conversion, we use the method specific to Büchi Automata on finite inputs described by Giannakopoulou and Havelund in [5], which is based on earlier work [4] on converting LTL to a form of Büchi Automaton for the purposes of model checking. This method results in NBA with finite-trace semantics.

2.3 Nondeterministic Büchi Automata

Nondeterministic Büchi Automata (NBA) extend nondeterministic finite automata (NFA) to operate on infinite-length words, allowing us to use a nondeterministic Büchi Automaton to represent the language of a Linear Temporal Logic statement [1].

A nondeterministic Büchi Automaton is a tuple $\langle Q, Q_0, F, \Sigma, T \rangle$, such that Q is a finite set of states, $Q_0 \subseteq Q$ is a set of start states, $F \subseteq Q$ is a set of accepting states, Σ is the input alphabet, and the transition function is $T : Q \times \Sigma \to 2^Q$.

We refer to Q hereafter as the *safety state* of the physical system. The alphabet Σ of a NBA consists of all possible physical configurations of the system. These NBA are modified from canonical NBA to accept finite traces.

2.4 Deterministic Büchi Automata

Runtime verification for stochastic systems as described in this paper requires a model of the safety requirements with a complete transition function, which a NBA does not guarantee. We obtain this function by converting the NBA of the safety requirements into a deterministic Büchi Automaton (DBA). A DBA is defined similarly to an NBA except that it may only have one start state $q_0 \in Q$, and the transition relation $T : Q \times \Sigma \to Q$ must be complete.

NBA on finite traces can be converted to an equivalent deterministic Büchi automaton without loss of expressiveness through subset or powerset construction. A method for doing so is described by Giannakopoulou in [5].

After conversion, a DBA contains a special state $q_\varnothing$ that denotes a violation of the safety requirements, and is the only non-SAFE state of the DBA.

3 The Probabilistic Hierarchical Constraint Automata Model

When safety properties are written over hidden system states, runtime verification of these properties requires a model of system behavior. We use the *Probabilistic Hierarchical Constraint Automaton* (PHCA) [14] formalism because it allows us to concisely and accurately model mixed hardware/software systems that degrade or fail, such as planetary rovers or cars. PHCA allow for probabilistic behavior, which is a reasonable model of random hardware failure.

PHCA are derived from HMMs. Like an HMM, a PHCA may have hidden states and transition probabilistically. Unlike an HMM, PHCA introduce the notion of constraints on states as well as a hierarchy of component automata. Systems are modeled as a set of individual PHCA components that communicate through shared variables. Discrete modes of operation representing nominal and faulty behavior are specified for each component. Components may transition between modes probabilistically or based on system commands. Additionally, modes and transitions may be constrained by the modes of other components.

For an example and more detail, the reader is referred to [9]. Note that another model providing the transition and observation probabilities required in Section 4.2 may be substituted, such as a less sophisticated HMM.

4 Runtime Verification for Stochastic Systems

Traditional runtime verification does safety monitoring of software systems in which state can be directly observed. In this section we extend the problem to that of safety monitoring of mixed hardware / software systems that can fail, and solve this problem by incorporating stochastic behavior and hidden state.

If it is assumed that the state of a mixed system is observable, then runtime verification may be used to monitor the safety of such a system. However, due to incomplete or faulty sensing, it is not realistic to assume that the state of an embedded system is generally observable. Therefore, in the case in which the system state x is hidden and $\mathbf{\Lambda}$ involves these hidden states, we estimate the safety of the system as a belief distribution, as described in Section 4.1. Section 4.2 derives an expression for this belief distribution in terms of system probabilities.

4.1 Extension to Hidden-State

Our system is drawn as a time-evolving graphical model in Figure 1.

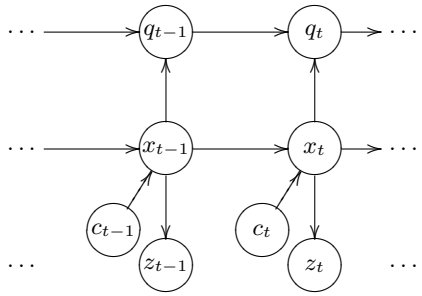

Fig. 1. A graphical model of an embedded system. The commands into the system are represented by c, observations z, physical system (hardware *and* software) state is x, and safety state is q. Subscripts denote time. Arrows denote conditional dependencies.

In this graphical model, q is the safety state of the system, defined as the state of the Deterministic Büchi Automaton (DBA) that describes a safety constraint on the system. Under the assumption that x is observable, the state q_t of the DBA at time t may be calculated from available information. However, when we remove the assumption that x is observable, q_t may no longer be directly calculated; the problem of safety monitoring can no longer be solved by runtime verification methods alone.

Instead, we want a capability that will evaluate the safety of the system given the available information: a safety specification $\mathbf{\Lambda}$, a plant model $\mathbf{\Phi}$, the control sequence $c_{1:t}$, and observation sequence $z_{1:t}$.[2] This capability estimates the *probability* that the system remains consistent with $\mathbf{\Lambda}$, that is, the probability that the system is SAFE. Let Q denote the set of states of the DBA for $\mathbf{\Lambda}$, and let Q_{SAFE} denote the set $Q/q_{\varnothing}$. That is, Q_{SAFE} is the set Q with the trap state $q_{\varnothing}$ removed. The probability $\mathbf{P}(\text{SAFE})$ is then equivalent to the probability of being in a SAFE state of the DBA at time t:[3]

$$\mathbf{P}(\text{SAFE}) = \mathbf{P}(q_t \in Q_{\text{SAFE}})$$

This probability can be derived from the probability distribution over states q of the DBA at time t, given the commands and observations, by summing over the SAFE states Q_{SAFE}:

$$\mathbf{P}(\text{SAFE}) = \sum_{q^j \in Q_{\text{SAFE}}} \mathbf{P}(q_t^j | z_{1:t}, c_{1:t}) \tag{1}$$

Thus the problem of stochastic safety monitoring of embedded systems reduces to the problem of finding the probability distribution over DBA states q, conditioned on the history of observations and commands. This probability distribution over q is often called a *belief state*, hence we abbreviate it as $\mathbf{B}(q_t)$.

4.2 Calculating Safety Belief

Let y_t represent the complete system state $< q_t, x_t >$ and let $\mathbf{B}(y_t)$ denote the belief over y at time t, that is $\mathbf{B}(y_t) = \mathbf{P}(q_t, x_t | z_{1:t}, c_{1:t})$. The graphical model in Figure 1, viewed in terms of y, is equivalent to a canonical hidden Markov model:

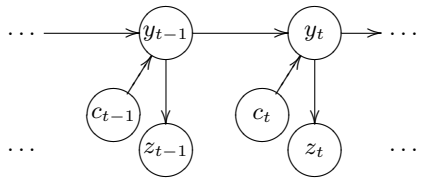

Fig. 2. Graphical model from Figure 1 with clustered state $y = q \otimes x$

The belief $\mathbf{B}(q_t)$ is obtained by marginalizing x_t out of $\mathbf{B}(y_t) = \mathbf{P}(q_t, x_t | z_{1:t}, c_{1:t})$:

$$\mathbf{B}(q_t) = \mathbf{P}(q_t | z_{1:t}, c_{1:t}) = \sum_{x_t} \mathbf{P}(q_t, x_t | z_{1:t}, c_{1:t}) \tag{2}$$

[2] Here subscripts denote time, hence $z_{1:t}$ is the vector of z's from time 1 to t.

[3] Summing over all states of the automaton except the trap state is necessary for the correct monitoring of liveness conditions.

and $\mathbf{B}(y_t) = \mathbf{P}(y_t|z_{1:t}, c_{1:t})$ is obtained through standard HMM filtering:

$$\mathbf{B}(y_t) = \eta \mathbf{P}(z_t|y_t) \sum_{y_{t-1}} \mathbf{P}(y_t|y_{t-1}, c_t)\, \mathbf{B}(y_{t-1}) \tag{3}$$

Equation (3) computes the belief state over the combined system state y, which can also be thought of as the combined DBA / PHCA state. To obtain a relation in terms of functions specified by these models, we manipulate (3) further by expanding y in the observation probability $\mathbf{P}(z_t|y_t)$ and the transition probability $\mathbf{P}(y_t|y_{t-1}, c_t)$, giving us (4). Applying the Chain Rule and simplifying based on conditional independence arguments yields (5):

$$\mathbf{B}(y_t) = \eta \mathbf{P}(z_t|q_t, x_t) \sum_{y_{t-1}} \mathbf{P}(q_t, x_t|q_{t-1}, x_{t-1}, c_t)\, \mathbf{B}(y_{t-1}) \tag{4}$$

$$= \eta\, \mathbf{P}(z_t|x_t) \sum_{y_{t-1}} \mathbf{P}(q_t|x_t, q_{t-1}) \mathbf{P}(x_t|x_{t-1}, c_t)\ \mathbf{B}(y_{t-1}) \tag{5}$$

Substituting Equation (5) into (2) produces the following, where η is a normalization constant:

$$\mathbf{B}(q_t) = \eta \sum_{x_t} \mathbf{P}(z_t|x_t) \sum_{y_{t-1}} \mathbf{P}(q_t|x_t, q_{t-1}) \mathbf{P}(x_t|x_{t-1}, c_t)\ \mathbf{B}(y_{t-1}) \tag{6}$$

Equation (6), which computes the belief state over the BA, is similar to the standard Forward algorithm for HMM belief state update (3). First, the next state is stochastically predicted based on each previous belief $\mathbf{B}(y_t)$ and on the transition probabilities of the models, then this prediction is corrected based on the observations received. An additional sum marginalizes out x_t, and the result is normalized by η. The observation probability $\mathbf{P}(z_t|x_t)$ and the transition probability $\mathbf{P}(x_t|x_{t-1}, c_t)$ are both functions of the model of the physical system. In an HMM, these are specified as a part of the model of the system. For PHCA, these system-wide probabilities must be calculated from the specified component transition and observation probabilities [8, 14]. The transition probability $\mathbf{P}(q_t|x_t, q_{t-1})$ in Equation (6) can be obtained from the transition function T_{D} of the specified deterministic Büchi Automaton as follows:

$$\mathbf{P}(q_t|x_t, q_{t-1}) = \begin{cases} 1 & \text{if } T_{\mathrm{D}}(q_{t-1}, x_t) = q_t \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The cost of computing (6) is entirely dependent on the sizes of Q and X. In order to find the probability of each q_t, we must loop twice over these sets. If n is the size of the combined set, $n = |Q \times X|$, then we have a time complexity of $O(n^2)$, and a space complexity of $O(n)$.

Finally, given the belief state determined by Equation (6), the probability that the system is currently SAFE is given by:

$$\mathbf{P}(\textsc{safe}) = \sum_{q_t \in Q_{\textsc{safe}}} \eta \sum_{x_t} \mathbf{P}(z_t|x_t) \sum_{y_{t-1}} \mathbf{P}(q_t|x_t, q_{t-1}) \mathbf{P}(x_t|x_{t-1}, c_t)\ \mathbf{B}(y_{t-1}) \tag{8}$$

5 Summary

In this paper we extended traditional runtime verification to deal with faulty mixed hardware/software systems by assuming a stochastic plant with hidden state, and then performing belief state update on the combined state of the deterministic Büchi Automata representing the safety requirements and the Probabilistic Hierarchical Constraint Automata representing the plant behavior. This method is innovative in its allowance for hidden state and probabilistic failure.

Preliminary validation has shown that our method is capable of quickly and accurately detecting safety violations on small models. Further work will seek to characterize the utility of these methods on larger models.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge, MA (2008)
2. Black, J.: System Safety as an Emergent Property. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA (April 2009)
3. Drusinsky, D.: The Temporal Rover and the ATG Rover. In: SPIN Model Checking and Software Verification. LNCS, vol. 1885, pp. 323–330. Springer (2000)
4. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: IFIP Conf. Proc. vol. 38, pp. 3–18 (1995)
5. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: 16th IEEE International Conference on Automated Software Engineering. San Diego, CA (2001)
6. Havelund, K., Roşu, G.: Java pathexplorer - a runtime verification tool. In: The 6^{th} International Symposium on AI, Robotics and Automation in Space (May 2001)
7. Kim, M., Kannan, S., Lee, I., Sokolsky, O., Viswanathan, M.: Java-MaC: a runtime assurance approach for Java programs. Formal Methods in System Design 24(2), 129–155 (March 2004)
8. Martin, O.B., Chung, S.H., Williams, B.C.: A tractable approach to probabilistically accurate mode estimation. In: Proc of iSAIRAS-05. Munich, Germany (September 2005)
9. Mikaelian, T., Williams, B.C., Sachenbacher, M.: Model-based monitoring and diagnosis of systems with software-extended behavior. In: AAAI-05. pp. 327–333. Pittsburgh, PA (July 2005)
10. Peters, D.K., Parnas, D.L.: Requirements-based monitors for real-time systems. IEEE Transactions on Software Engineering 28(2) (February 2002)
11. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (FOCS 1997). pp. 46–57 (1977)
12. Sammapun, U., Sokolsky, O., Lee, I., Regehr, J.: Statistical runtime checking of probabilistic properties. In: Proceedings of the 7th International Workshop on Runtime Verification (RV 2007). LNCS, vol. 4839, pp. 164–175. Springer (2007)
13. Sistla, A.P., Srinivas, A.R.: Monitoring temporal properties of stochastic systems. In: Proc of 9th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2008). pp. 294–308 (2008)
14. Williams, B., Chung, S., Gupta, V.: Mode estimation of model-based programs: Monitoring systems with complex behavior. In: Proc of the International Joint Conference on Artificial Intelligence. pp. 579–585. Seattle, WA (2001)