Verification of Adaptive Systems

Laura Pullum[1] and Xiaohui Cui[2]
Oak Ridge National Laboratory, Oak Ridge, TN 37830

Emil Vassev[3] and Mike Hinchey[4]
Lero – The Irish Software Engineering Research Centre, Limerick, Ireland

and

Christopher Rouff[5] and Richard Buskens[6]
Lockheed Martin Advanced Technology Laboratories, Cherry Hill, NJ 08002

Adaptive systems are critical for future space and other unmanned and intelligent systems. Verification of these systems is also critical for their use in systems with potential harm to human life or with large financial investments. Due to their nondeterministic nature and extremely large state space, current methods for verification of software systems are not adequate to provide a high level of assurance for them. The combination of stabilization science, high performance computing simulations, compositional verification and traditional verification techniques, plus operational monitors, provides a complete approach to verification and deployment of adaptive systems that has not been used before. This paper gives an overview of this approach.

Nomenclature

a	=	formal specification
α	=	terminal state-reachable set of states
A	=	software system
Act	=	set of actions
AP	=	atomic propositions
Cm	=	confidence measure
g	=	state graph
HPC	=	high performance computing
L	=	labeling function
LTS	=	labeled transition system
p	=	correctness properties
R	=	possible state transitions
S	=	possible system states
S_0	=	set of initial states
τ	=	transitions
TL	=	temporal logic
ω	=	set of terminal states
x	=	number of inputs
y	=	number of optimal samples

[1] Sr. Research Scientist, Computational Data Analytics Group, P.O. Box 2008, MS-6085, AIAA Member
[2] Joint Research Scientist, Computational Data Analytics Group, ORNL; Computer Science Department, New York Institute of Technology, 1855 Broadway, Manhattan, NY 10023.
[3] Research Fellow, Lero—the Irish Software Engineering Research Centre, University of Limerick, T2-023
[4] Director, Lero—the Irish Software Engineering Research Centre, University of Limerick, T2-013
[5] Engineering Manager, Informatics Laboratory, 3 Executive Campus, Suite 600, Senior Member of AIAA
[6] Engineering Manager, Applied Science Laboratory, 3 Executive Campus, Suite 600

I. Introduction

To solve future complex mission needs in space exploration and aeronautics science, NASA roadmaps[1] and Space Technology Grand Challenges[2] have identified the need for adaptive systems—systems that autonomously adjust their behavior during operation due to unanticipated events, changes in environment, etc. Adaptive systems will be critical in future missions because many of them will take place in parts of the solar system where manned missions are simply not possible, and to where the round-trip delay for communications to spacecraft can exceed 40 or more minutes. This means that the decisions on responses to exploration opportunities as well as problems and undesirable situations that arise must be made *in situ* rather than from people or systems at ground control on Earth. To handle these unforeseen situations, intelligence and adaptation will need to be added to the missions both to individual spacecraft and collectively in teams of autonomous systems. In addition to the time lag in communications, adaptive systems will also be needed in systems where there is limited or no oversight of the system operation[3]. This could be at remote science posts, autonomous unmanned aerial vehicles (UAVs) or other unmanned systems that are operating in dynamic environments.

Complete testing of these systems will be nearly (if not completely impossible) since the adaptations can be difficult or impossible to forecast and because there are may be limited to no opportunities to effectively monitor and adjust such systems during operation (especially for deep space missions). Therefore, system and software verification will be critical to these systems correct operation after deployment[4].

Verification of these systems using traditional verification techniques is not adequate. Due to the nature of these systems' need to adjust/adapt their behavior on the fly, the state space of the systems can be astronomical. Aggressive state space reduction is required for modern automated verification techniques to work. Unfortunately, this leads to the need to reduce the granularity of the system models, which results in low-precision models that no longer adequately represent the original system.

To address these verification issues for adaptive systems, we are developing Adaptive Verification (*AdaptiV*), a tool chain and methodology (Figure 1) for verifying adaptive systems that alleviates the above challenges. AdaptiV consists of the following parts:

1) a stability analysis capability that identifies instabilities given a system model and partitions the system model into stable and unstable component models;
2) a state space reduction capability that prunes the state space of an unstable component model without loss of critical fidelity;
3) high performance computing (HPC) simulations to explore component behavior over a wide range of an unstable component's reduced state space and produce a statistical verification for the component;
4) a compositional verification capability that aggregates individual component verifications; and
5) operational monitors to detect and take action to correct undesired unstable behavior of the system during operation.

The remainder of this paper gives additional background on the challenges of verification of adaptive systems and discusses how the above 5 parts work to address verification of adaptive systems.

II. Background

Contemporary software systems have massive state spaces. This is particularly true for adaptive systems, where components of such systems operate concurrently, interact with each other and the environment, and react in response to changes in the environment. The huge state spaces are the result of the combinatorial explosion caused by non-deterministic interaction (interleaving) of component operations and the environment. Typical modern automated verification techniques, such as automated theorem proving and model checking, do not scale to support such large state spaces. For these techniques to be effective, the state space of the targeted systems must be substantially reduced. State space reduction is achieved by aggregating state transitions into an abstract (coarser-grained) finite state model of the system. The technique effectively reduces the total number of states to be considered, but also reduces the fidelity of the system model. The key is that the abstract model must remain precise enough to adequately represent the original system in dimensions of interest. Thus, a tradeoff exists between the size and precision of the models. Today, only very abstract, low fidelity models can be automatically verified. What's needed for adaptive systems are techniques to support automated verification of a much larger state space.

Stabilization science has been used to verify the stability of a trained neural network, which is one form of an adaptive system[5-9]. AdaptiV is building on this body of work to verify a broader range of adaptive systems. It is using stability analysis to identify the unstable parts of an adaptive system. These parts will be further analyzed

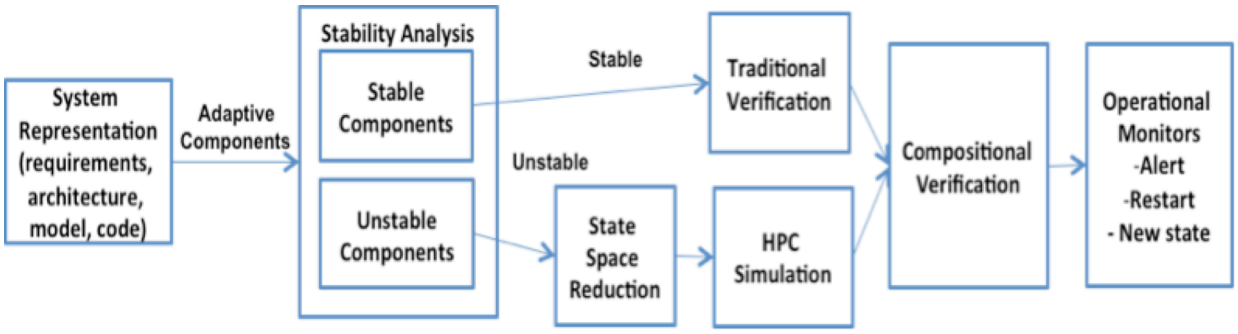

Figure 1. Process for verification of adaptive systems

using HPC simulation over a large number of runs to compute a confidence level in their ability to converge around a stable operating point or region. While adaptive systems may be inherently unstable because of operational needs – e.g., the need to adapt in real time – this is not necessarily a reason for failing verification. An unstable system may still converge, even though complete verification may not be possible.

The above results will then be combined to yield a probabilistic measure of confidence in component behavior and provide state space convergence parameters that identify potential symptoms of unstable behavior. Where comprehensive verification is not possible, operational monitors can be deployed with the adaptive system. Monitors will be able to be automatically generated and deployed to detect non-convergence symptoms during operation and guide the adaptation towards stable behavior.

III. Model Checking

A promising, and lately popular, technique for software verification is model checking[10]. This approach advocates *formal verification tools* whereby software programs are automatically checked for specific design flaws by considering *correctness properties* expressed in a *temporal logic* (TL). A temporal logic augments propositional logic with modal and fix-point operators. In general, model checking provides an automated method for verifying finite state systems by relying on efficient graph-search algorithms. The latter help to determine whether or not system behavior described with temporal correctness properties holds for the system's state graph.

A general model-checking problem is: given a software system $\boldsymbol{A}$ and its formal specification $\boldsymbol{a}$, determine in the system's state graph $\boldsymbol{g}$ whether or not the behavior of $\boldsymbol{A}$, expressed with the correctness properties $\boldsymbol{p}$, meets the specification $\boldsymbol{a}$. Formally, this can be presented as a triple ($\boldsymbol{a}$; $\boldsymbol{p}$; $\boldsymbol{g}$). Note that $\boldsymbol{g}$ is the state graph constructed from the formal specification in a *labeled transition system* (LTS)[8] format. Formally, an LTS can be presented as a Kripke Structure[8], which is a tuple ($\boldsymbol{S}$; $\boldsymbol{S_0}$; $\boldsymbol{Act}$; $\boldsymbol{R}$; $\boldsymbol{AP}$; $\boldsymbol{L}$) where: $\boldsymbol{S}$ is the set of all possible system states; $\boldsymbol{S_0 \subseteq S}$ is a set of initial states; $\boldsymbol{Act}$ is the set of actions; $\boldsymbol{R \subseteq S \times Act \times S}$ are the possible state transitions; $\boldsymbol{AP}$ is a set of special *atomic propositions*; $\boldsymbol{L : S \rightarrow 2^{AP}}$ is a labeling function relating a set $\boldsymbol{L(s) \in 2^{AP}}$ of atomic propositions to any state $\boldsymbol{s}$, i.e., a set of atomic propositions true in that state. Note that in order to make an LTS appropriate for model checking, each state $\boldsymbol{s}$ must be associated with a set of atomic propositions $\boldsymbol{AP}$ true in that state. Therefore, if we turn a software system into a state machine, we can use temporal logics to express temporal correctness properties of the program. The most common such property is the *global invariant*, i.e., a property that holds in all reachable states of a state machine, or equivalently, always holds during the execution of a program.

The biggest issue model checking is facing today is the so-called *state explosion problem*[10]. In general, the size of a state graph is at least exponential in the number of tiers running as concurrent processes, because the state space of the entire system is built as the Cartesian product of the local state of the concurrent processes. To overcome this problem, modern model checking tools strive to reduce the state space of the targeted software systems.

Note that a straightforward model of a contemporary concurrent software system has a large and complicated state space and reduction is an important technique for reducing the size of that state space by aggregating state transitions into coarser-grained state transitions. State-space reduction is achieved by constructing an abstract (coarser-grained) finite state model of the system, which eventually is still powerful enough to verify properties of interest. The technique effectively reduces the total amount of states to be considered but is likely to reduce the granularity of the system to a point where it no longer adequately represents that system. The problem is that although the abstract model is relatively small it should also be precise to adequately represent the original system. The latter requirement tends to make the abstract models large, because the size of a transition system is exponential in the number of variables, concurrent components and communication channels. However, large models make

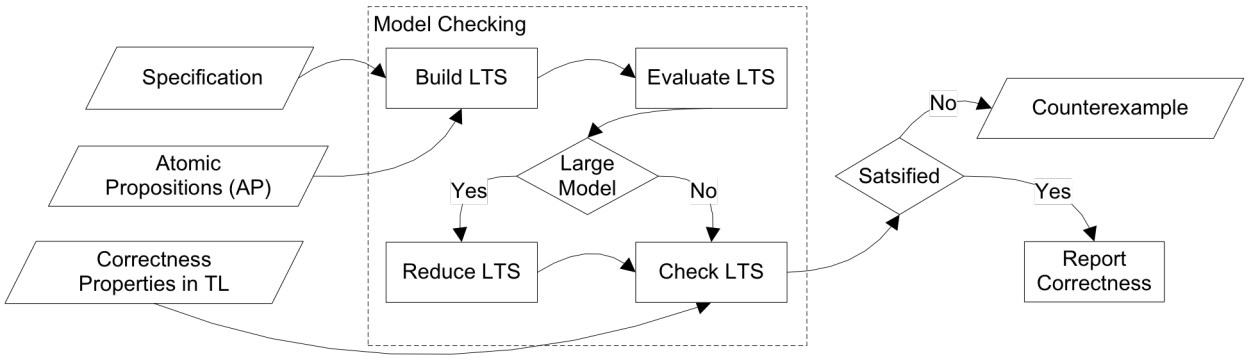

Figure 2. The Model-Checking Approach

automated verification extremely inefficient, thus introducing tradeoffs between the size and precision of the models which considerably reduces their effectiveness.

Figure 2 depicts a generic view of the model-checking verification method. Note that in the case that a correctness property is not satisfied, the method returns a *counterexample*. The latter is an execution path of the LTS for which the desired correctness property is not true. If model checking has been performed on the entire LTS, then the property does not hold for the original system specification. Otherwise, in the case that a reduced LTS has been used (state-explosion techniques have been applied), the information provided by the counterexample is then used to refine the reduced model. Numerous formal tools allowing verification by model-checking have been developed, such as Spin, Emc, Tav, Mec, XTL, etc. Despite best efforts and the fact that model checking has proved to be a revolutionary advance in addressing the correctness of critical systems, software assurance for large and highly-complex software is still a tedious task. The reason is that high complexity is a source of software failures, and standard model checking approaches do not scale to handle large systems very well due to the state-explosion problem.

Model checking is the most prominent automatic verification mechanism today. However it requires finite state models in order to perform automatic verification of all the possible execution paths of a system. However, the adaptive systems (or individual adaptive components) are *intrinsically non-deterministic*, which means that they may have a huge state space. As a result, abstract models needed by model checking are difficult to set up and to use. Hence, validation by using model checking is possible neither for the whole system nor for the individual adaptive components. In such a case, to perform limited model checking on some of the components, we need to determine the non-adaptive and adaptive (unstable) parts of an adaptive system.

IV. Stabilization Science

Stabilization science[11] provides a common approach to studying system stability, where a system is linearized around its operating point to determine a small-signal linearized model of that operating point. The stability of the system is then determined using linear system stability analysis methods such as Routh-Hurwitz, Root Locus, Bode Plot, and Nyquist Criterion. AdaptiV will use stabilization science on a model of an adaptive system to partition it into a collection of stable (deterministic) and unstable (non-deterministic) components, apply traditional techniques to verify the stable components, apply high performance computing simulation to explore the state space of unstable components, compute a verification confidence for each component, and use compositional verification techniques to produce an overall verification and verification confidence for the whole system.

Identifying the unstable parts of an adaptive system is key to our verification approach. The unstable parts introduce uncertainty in system behavior where, in contrast, a stable system transits from one safe state to another safe state. Currently, there is no efficient way to determine the overall stability of a complex concurrent system, such as spacecraft software. Due to the state space explosion problem, a system-level stability check may suggest divergent behavior since over an infinite state space there may be an infinite sequence of successively weaker assertions, none of which is stable. To address this problem, we are using stabilization science to model an adaptive system and analyze it to identify and partition the model into a collection of stable and unstable components. We are using the results of the stability analysis to create small-signal linearized models for all the system components. We anticipate that the linearized models of system components will yield a relatively small state space, enabling

their effective analysis. Automatic stability analysis of the components might be best performed via time domain simulation using small-signal models. It should be noted that the lack of unstable components does not automatically guarantee system stability and compositional verification will need to be performed to ensure the desired system behavior.

The main challenge in building linearized models of concurrent systems, such as spacecraft, is dealing with interference, i.e., the possibility that concurrent processes may concurrently make changes to shared resources, e.g., memory. A possible solution to the problem is defining a *synchronization scheme* to avoid interference. Most of such schemes rely on some form of access denial, such as locks. Though locks make it easy to reason about correctness, they may also cause loss of efficiency as processes wait to acquire access to needed resources. Therefore, a locking scheme may become increasingly fine-grained, attempting to deny access to the smallest possible size of resource, to minimize waiting and maximize concurrency. A linearized model must implicitly or explicitly take the actions of other processes into account. Broadly speaking, a linearized model of an operating point should encode the environment and the concurrent processes into the proof: all pre- and post-conditions must be shown to be unaffected (or stable) under the actions of the environment or the other concurrent processes. Once non-interference has been established, the proof can be carried forward exactly as for a sequential program. Note that automatically ensuring such non-interference can be problematic in many cases.

Partitioning the system into components, verifying each component, then using compositional techniques to provide an overall verification for the system is not new. What is unique is the application of stabilization science to partition the system into stable and unstable components. Stable components represent deterministic or non-adaptive behavior and can be verified using traditional techniques. Unstable components–those representing non-deterministic or adaptive behavior – require state space exploration beyond that which can be achieved using traditional techniques.

V. State Space Reduction

Stable components identified during the stability analysis represent deterministic or non-adaptive behavior. These components will be verified using traditional techniques. Unstable components may require state space exploration beyond that which can be achieved using traditional techniques. For these components, we are:

1) Pruning the state space by identifying isomorphic elements in the state space.
2) Examine patterns in the state space (using clustering, categorization, or other pattern identification approaches) to further reduce the state space.

As needed, we will examine other ways to reduce the state space in ways that provide sufficient confidence that model behavior appropriately represents actual system behavior.

VI. High Performance Computing

Stability analysis methods perform exhaustive exploration of all possible behaviors. Partitioning the system into stable and unstable components will reduce the size of the state space requiring exploration and will help to speed up the exploration of the remaining state space. In spite of this, we anticipate that it will still be impossible to explore the entire state space of a large adaptive system with limited memory resources and limited time.

To reduce memory, we will take a lightweight snapshot of an unstable component's state – consisting of a state's signature (a hash compaction of an actual state) and the trace (the sequence of return values from its path decision function). To restore the component's state, AdaptiV will replay the sequence of choices from the initial state. However, reconstructing states is a slow and CPU-intensive process, especially when traces are deeper.

To reduce runtime, we will use HPC to determine if and how the unstable components found during stability analysis will converge during adaptation. Parallelizing simulations will allow multiple state space explorations to occur simultaneously. We plan to investigate the use of HPC to achieve exhaustive exploration on the unstable components. All HPC "nodes" (or processing elements) will reconstruct and clone the states from their traces concurrently and explore them on different nodes. Checkpoints of actual component states on one node can be efficiently distributed to other nodes, through live operating system processes that use thin virtualization techniques. In addition, such techniques facilitate the use of distributed hash tables, treating the lightweight snapshot of the states as network objects to achieve fair load balancing and reduce the network communication for status exchange between the divided components.

As we indicated previously, even with the help of HPC, we do not anticipate that any computational model will ever be fully verified, given limited memory and time resources. To overcome this limitation, AdaptiV will provide

a percentage of confidence level or confidence measure. The basic confidence measure will be calculated by following equation:

$$Cm = x*(2*0.5^{y})$$

where *Cm* is the confidence level measure, *x* is the total number of inputs and *y* is number of optimal samples. How to optimize the sample results to maximize coverage of the state space is an open research question that will be explored on this project. How to integrate the confidence measures of the unstable components for generating the whole complex system's confidence measure needs further stochastic research. Even so, AdaptiV can increase the statistical confidence level beyond that of traditional model checking tools.

VII. Compositional Verification

Adaptation significantly complicates system design because adaptation of one component may affect the quality of its provided services, which may in turn cause adaptations in other components. The mutual interaction among system components affects overall system behavior. Hence, it is not sufficient to verify each component separately to ensure system correctness. What's needed is an ability to check the adaptation process as a whole. This is a complex and error-prone task. In our approach, we will apply compositional verification[12,13] techniques to produce an overall system-wide verification. We will consider combinations that characterize important invariants, classified into: mission goal invariants, behavior invariants, interaction invariants and resource invariants. Here, behavior invariants are over-approximations of components' reachability of safe states, and interaction invariants are global constraints on the states of components involved in interactions. Selecting the most appropriate set of invariants and determining heuristics for computing invariants (e.g., interaction invariants) are major difficulties in designing a compositional verification technique for adaptive systems. We explore this selection process as part of the ongoing research.

A set of possible abstraction rules for generating the compositional model is provided in[14,15]. These seven conflict-preserving abstraction rules will be analyzed for applicability to compositional verification for adaptive systems. Let α and ω be propositions as follows: α specifies a set of states from which terminal states are required to be reachable, and ω represents a set of terminal states. Initial abstraction rules[13] to investigate involve:

- Observation equivalence
- Removal of α-markings
- Removal of ω-markings
- Removal of non-coreachable states
- Determination of non-α states
- Removal of τ-transitions leading to non-α states
- Removal of τ-transitions originating from non-α states

While compositional verification alone cannot guarantee complete correctness of an adaptive system, it can prove such things as deadlock-freedom and overall mission-goal reachability.

VIII. Operational Monitors

Because an adaptive system cannot be completely verified, even with the proposed above approach, operational monitors should be deployed with the end system that monitor the adaptations. These monitors would be based on the results of the stability analysis and the HPC simulations. They would monitor either all or only those adaptive components that had trouble converging during the simulations. The monitors can provide alerts that the system is not converging during an adaptation within a given time, restart components of the system that are not converging, or force the system into a known state if any adaptations do not converge within a specified amount of time. Alerts could be adjusted by ground control based on the severity, known issues, etc. Software patches could be made during a mission based on convergence issues (which could be different from those found during simulations), reduce the amount of adaptation, or even increase the amount of adaptation if it is going well.

Instead of one large monitor, we anticipate that it will be more advantageous from a system efficiency stand point to deploy multiple monitors with the end system—one or more for each adaptive component. To reduce overhead processing, the monitors would only have to operate when an adaptive component is executing; otherwise, they could remain dormant.

The monitors would be configured with information from the HPC simulations regarding convergence times for an adaptive component during adaptation. These times would provide bounds for how long an adaptive component may take to converge. When the convergence is outside these bounds, appropriate action, as noted above, could be taken. In addition, end states (variable values, etc.) that indicate that adaptation has completed would also be used by the monitors. When these end states are reached the monitor would know that the adaptation has converged successfully and that the monitor can do any necessary reporting and go back to sleep.

IX. System Inputs

The inputs to the AdaptiV will consist of a model of the adaptive system. The model would be derived from either the system requirements or the system design. In the beginning we anticipate that this conversion will be a manual one, but there are tools that could be used as a starting point that could automatically produce the needed model. This could range from modified UML tools or other systems specification and verification tools. The type and structure of the model used for the stability analysis will depend on the type of stability analysis used (this is an area of research for the project). The parts of the adaptive system that are determined to be stable could be verified using the same techniques as the non-adaptive components of the system, which are more traditional techniques. We are currently concentrating on the unstable parts of the system so the verification of the components that are deemed stable are outside our current research thrust.

X. Conclusion

From the NASA roadmaps[1] and Space Technology Grand Challenges[2], it is clear that the use of adaptive systems will be important for future space systems and missions as well as other life critical systems. Due to their large state space, non-determinism, and the changing nature of these systems, traditional verification techniques are not adequate. The combination of stabilization science, HPC simulations, compositional verification and traditional verification techniques, plus operational monitors, provides a complete approach to verification and deployment of adaptive systems that has not been used before. This paper discussed the components of such a system and how these components would work together. The stabilization system would check for unstable adaptive components of an adaptive system. The stable components would be verified using traditional verification techniques. The unstable components would have their state space reduced and then simulated using high performance computing to determine whether they converge. Components that do not converge in a given time period would need to be redesigned. Unstable components that do converge would be deemed verified and then would be verified with the stable components using compositional verification techniques. Operational monitors would also be deployed with the system to monitor adaptations to make sure they converge. If they do not converge in a predetermined amount of time the monitors would stop the adaptation and put the system back into a known state.

Acknowledgments

The research described in this paper was conducted in part at ORNL (managed by UT-Battelle). This manuscript has been co-authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

This work was supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero–the Irish Software Engineering Research Centre.

References

[1]NASA, Space Technology Roadmaps, Office of the Chief Technologist, October 2011, http://www.nasa.gov/offices/oct/home/roadmaps/index.html

[2]NASA, Space Technology Grand Challenges, Office of the Chief Technologist, October 2011, http://www.nasa.gov/offices/oct/strategic_integration/grand_challenges_detail.html

[3]Vassev. E., Hinchey, M., Rouff, C., and Sterritt, R. Building Resilient Space Exploration Systems. IEEE IT Professional. April, 2012.

[4]Hinchey, M., Rash, J., Truszkowski, W., Rouff, C. and Sterritt, R. You Can't Get There from Here! Large Problems and Potential Solutions in Developing New Classes of Complex Computer Systems. In Conquering Complexity. Eds. Hinchey, M. and Coyle, L. Springer 2012.

[5]Taylor, B.J., Darrah, M.A., Pullum, L.L., et al., Methods and Procedures for the Verification and Validation of Neural Networks, Brian Taylor, ed., Springer-Verlag, 2005.
[6]Pullum, L., Taylor, B., and Darrah, M. Guidance for the Verification and Validation of Neural Networks, IEEE Computer Society Press (Wiley), 2007.
[7]Pullum, L.L., Darrah, M.A., and Taylor, B.J. Independent Verification and Validation of Neural Networks - Developing Practitioner Assistance, Software Tech News, July, 2004.
[8]Yerramalla, S., Fuller, E., Mladenovski, M., Cukic, B. Lyapunov Analysis of Neural Network Stability in an Adaptive Flight Control System. Self-Stabilizing Systems 2003: 77-91.
[9]Phattanasri, P., Loparo, K.A., Soares, F. Verification and Validation of Complex Adaptive Systems. EECS Department, Case Western Reserve Univ., Contek Research, Inc., April 2005.
[10]Baier, C. and Katoen, J.-P., Principles of Model Checking, MIT Press, 2008.
[11]Emadi, A. and Ehsani, M., Aircraft Power Systems: Technology, State of the Art, and Future Trends, Aerospace and Electronic Systems Magazine, IEEE, Volume 15, Issue 1, Jan. 2000, pp. 28 - 32.
[12]Roever, W.-P. de, Boer, F. de, Hanneman, U., Hooman, J., Lakhnech, Y., Poel, M., and Zwiers, J., Concurrency Verification: Introduction to Compositional and Non-compositional Methods, Cambridge University Press, 2001.
[13]Francis, R., An implementation of a compositional approach for verifying generalized nonblocking. Working Paper: 04/2011, The University of Waikato, Department of Computer Science, Hamilton, NZ.
[14]Leduc, R., and Malik, R. (2009). Seven abstraction rules preserving generalized nonblocking. Working Paper: 07/2009, The University of Waikato, Department of Computer Science, Hamilton, NZ.
[15]Leduc, R., and Malik, R. (2009). A compositional approach for verifying generalized nonblocking. Proceedings of 7th International Conference on Control and Automation, ICCA '09, (pp. 448-453). Christchurch, NZ.