Digital System Robustness via Design Constraints: The Lesson of Formal Methods

Jackson R. Mayo*, Robert C. Armstrong†, Geoffrey C. Hulette‡
Sandia National Laboratories
P.O. Box 969
Livermore, California 94551-0969, USA
Email: *jmayo@sandia.gov, †rob@sandia.gov, ‡ghulett@sandia.gov

***Abstract*—Current programming languages and programming models make it easy to create software and hardware systems that fulfill an intended function but also leave such systems open to unintended function and vulnerabilities. Software engineering and code hygiene may make systems incrementally safer, but do not produce the wholesale change necessary for secure systems from the outset. Yet there exists an approach with impressive results: We cite recent examples showing that formal methods, coupled with formally informed digital design, have produced objectively more robust code even beyond the properties directly proven. Though discovery of zero-day vulnerabilities is almost always a surprise and powerful tools like semantic fuzzers can cover a larger search space of vulnerabilities than a developer can conceive of, formal models seem to produce robustness of a higher qualitative order than traditionally developed digital systems. Because the claim is necessarily a qualitative one, we illustrate similar results with an idealized programming language in the form of Boolean networks where we have control of parameters related to stability and adaptability. We argue that verifiability with formal methods is an instance of broader design constraints that promote robustness. We draw analogies to real-world programming models and languages that can be mathematically reasoned about in contrast to ones that are essentially undecidable.**

***Index Terms*—Digital design; programming models; robustness; security; formal methods; complex systems**

I. Introduction

Digital hardware and software are effectively undecidable in the general case due to their complexity [1], [2]. That is, there exists no practical means for exhaustively answering questions about the behavior space of an *arbitrary* circuit or program to ensure that its design meets requirements. Because the positive function of a digital system originates from human intent, functional behaviors are of relatively low complexity and are testable. However, the safety and security requirements for what a digital system must *not* do are notoriously difficult to verify. Indeed, "zero-day" (freshly discovered) vulnerabilities triggering unanticipated behavior are routinely found in hardware and software after deployment, despite strong efforts at testing and code hygiene during development.

The digital design and verification approach with the greatest empirical success in preventing unanticipated behavior is the use of formal methods [3] – applying automated logical reasoning to exhaustively analyze a mathematical model of the design. Consistent with the general principle of undecidability, the design must be suitably *constrained* to be analyzable by this approach. However, the mathematical guarantees provided by formal methods tools are limited to the specific requirements or assertions that are encoded by the developer in the formal modeling language and that are tractable for the tools to verify. While these formal assertions can be expansive in covering vast combinatorial behavior spaces that would be infeasible to cover by testing, the assertions directly encode only those characteristics of undesired behavior that the developer can foresee and express mathematically. Yet, it is observed that some complex systems show robustness against a wide range of unanticipated behaviors well beyond those for which formal guarantees were obtained.

Arguably the greatest feat of the 20th century is the development of digital electronics, computing the mathematics of intricate logic reliably. Since all mathematics is self-evident, one would expect that if the computation is rendered perfectly there could be no surprises in what a digital system is capable of. Because of the complexity of digital systems, this is not so. In most cases digital systems, like complex systems more generally, have only limited mathematical basis for their modeling and design. Complex systems are typically modeled as a large set of entities evolving in time and a network of connecting bonds that transmit information between entities, affecting their evolution. Abstract networks of this sort are commonly used to simulate power distribution grids, computers and networks of computers, and economic and social systems. Though they have increasing application in industry and the military, complex system models differ from those traditionally studied in science. Complex systems are generally not amenable to the conventional divide-and-conquer *reductionist* approach, i.e., they are neither reducible to a tractable phenomenological equation, nor reducible to a traditional statistical-mechanics treatment. Complex systems are the result of a scalably large number of nonlinear interactions that, acting together, produce "emergent" behaviors that cannot be easily deduced from the constituent parts. Very large-scale systems in everyday use, such as the electric power grid and the Internet, have emergent behavior that is usually beneficial, but in some cases detrimental (growth of a botnet on the Internet) or catastrophic (a widespread blackout of the electric grid). Both robustness to common failure modes and brittleness to rare unexpected events are characteristic of complex systems.

Here we argue that formally informed design of complex

978-1-4799-5927-3/15/$31.00 ©2015 IEEE

digital systems, ostensibly for the verification of safety, liveness, etc., has an ancillary effect of conferring general robustness beyond what is mathematically guaranteed. We maintain that verifiability with formal methods is an instance of broader design constraints that promote robustness. Thus, a formal methods approach is useful beyond the specific assertions that are exhaustively verified, and is complementary with more general principles of stability and information damping drawn from, e.g., biological and social complex systems. This suggests that the insights of formal methods can be extended in power and scalability as part of designed-in robustness of the system as a whole.

II. Limitations of Formal Methods: Semantics and Tractability

Formally proven software and hardware are only "proven" for explicit assertions made ahead of time, during the development process. The most surprising and arguably damaging vulnerabilities are so-called zero-day vulnerabilities, unrelated to previous faults and largely unguessable. At first blush, it would be surprising that a formal approach would be effective in making digital systems robust to unforeseen *types* of faults. Safety, security, liveness, etc., requirements can be expressed and proven broadly, but the inventiveness of exploits and the unpredictability of faults are notorious. High-consequence vulnerabilities can lie dormant in even ubiquitously used code [4] for decades before being "discovered". It seems unreasonable to expect that design of hardware and software, even formally proven design, would be able to anticipate vulnerabilities of such byzantine complexity.

Formal analysis tools are generally available in two forms:

1) *Theorem provers* provide an environment in which the designer can prove requirements manually with general logical reasoning. Automation is limited and the process is labor intensive, but the format is general, admitting virtually any level of semantics. When analysis is complete, an independently checkable "proof term" is produced.
2) *Model checkers* seek to exhaustively check design requirements against all of the states in the design that are accessible, given the initial conditions. Partial order reduction [5] and other tricks are used to make the computation tractable. In general, no proof term is produced and the designer has to take the tool's word for it that a proof of the requirements has been accomplished.

Model checkers are more automated and require less training to use effectively, but theorem provers are considered to be more powerful. Crucially, in both theorem provers and model checkers, programs are constrained to be written in a modeling language that expresses the logic and semantic constraints of what the tools can analyze. Some commercial model checkers use a native programming language as the modeling language but are limited in the requirements they can prove (e.g., Cadence [6] uses Verilog). Usually, to be able to reason about program requirements effectively, these modeling languages are necessarily more restrictive than a general programming language like C or Python. In many cases, code in a general programming language can be synthesized from the model code. As a general rule, more powerful requirements can be proven with modeling languages that are more constrained, but which are necessarily more difficult to program.

Another limitation of the formal approach is that requirements are valid only in the semantics in which they were proven. The modeling language defines those semantics, and if the program expressed in the modeling language is translated into an implementation, then it falls heir to any vulnerabilities in the new semantics. For example, timing vulnerabilities in hardware rely on the particular layout in silicon that, in effect, change the logic of the original design.

Despite these limitations, there is experimental evidence (see Section III) that the process of formal verification incorporated into the design process not only ensures that known and specified requirements are met, but also enhances the design's robustness against unknown vulnerabilities.

III. The Surprising Power of a Formal Design Approach

As mentioned previously, systems that are designed to be amenable to formal verification appear to exhibit enhanced robustness, even beyond the properties for which they are verified. We speculate that this "robustness for free" is attributable to the restricted set of possible behaviors that result from using programming languages, models, and design idioms that are crafted specifically to be analyzed. To simplify potential analyses, these tools restrict the programming model. The Ivory [7] embedded domain-specific programming language, for example, does not allow potentially unsafe type casts and ensures that pointers are not nullable, among other safeguards. In the SMACCMPilot project, an unmanned aerial vehicle (UAV, i.e., a drone) was designed with its embedded control software written in Ivory, and was dubbed "unhackable" after being subjected to extensive red team exercises [8]. This was in spite of the fact that very little of the UAV code was verified beyond the type safety properties enforced by Ivory itself. By comparison, commercial drones designed with conventional languages are notoriously susceptible to cyber attack [9].

The Compcert C compiler [10] also seems to exhibit this kind of robustness. Compcert is written in Coq, a programming language restricted in such a way that it can also serve as a sound proof theory. When researchers subjected a set of C compilers (including Compcert and mainstream compilers like GCC and Clang) to a large battery of randomly generated correctness tests, Compcert exhibited only a few errors in the parser and back-end code generation, and no errors at all in its verified core. The other, unverified compilers exhibited hundreds of errors in these "fuzzing" tests [11]. It seems likely that the randomized tests would eventually exercise unverified properties of Compcert's core, and yet empirically, the tests still failed to expose any errors.

It is important to note that our hypothesis does not predict increased robustness in systems where formal verification was done *after the fact*. The key is that *design for analysis* yields

increased robustness as a by-product, independent of when or even whether the analysis is performed. It should not and does not appear to apply to systems that were not designed for analysis but that were subjected to it after the fact. The LLVM compiler, for example, has undergone formal analysis for certain transformations [12], but was not designed to accommodate formal analysis. Therefore, we would not expect that analysis to translate to increased overall robustness, and indeed the fuzzing results [11] suggest that LLVM is no more robust than other mainstream compilers.

While we have concentrated on the unexpected benefits that formally analyzable languages and constructs confer, evidence [3] suggests that just high-level formal modeling of the requirements without any automated connection to the implementation code helps guide the programmer or put the programmer in a proper "frame of mind" to create a robust implementation. There is empirical evidence to suggest that application of formal modeling early in the planning stages of the development process yields outsize benefits later on, even absent any further formalization or verification [13], [14]. This evidence supports our hypothesis that even some limited consideration of analysis *up front* increases the robustness of the system.

IV. Design Constraints Promoting Analyzability and Robustness

The complexity of modern computer systems makes them difficult to design and analyze, and leads to pervasive reliability and security problems. A framework for understanding this behavior is to view computers in their most basic representation: as dynamical systems. For example, the low-level behavior of typical computer software and hardware has been found empirically to exhibit chaotic dynamical system features [15]. The use of formal methods, by contrast, enforces a *tractable* design semantics in which behavior spaces can be bounded and pruned in an exact manner.

Highly engineered and highly evolved complex systems in diverse domains, resulting from selection-driven adaptation, exhibit characteristic "edge of chaos" dynamics in which the response to perturbations is bounded probabilistically [16], [17]. Indeed, for a dynamical system to be a good computational device, it cannot be strongly overdamped and thus inert to any input, nor can it be so profoundly chaotic that the output is random to any input. To be useful, a computer must be close to the edge of chaos, also known as the critical manifold. The associated adaptation process is responsible for the robust functionality of biological organisms – even in the absence of any direct imposition of formal methods. While the adaptation process itself is optimized on the edge of chaos, there are reasons to seek a result that is somewhat subcritical, i.e., damps perturbations to some degree. Subcriticality generalizes the constraints imposed by formal analyzability – reducing the dimension of the effective design space, making some perturbations irrelevant, and making the system more likely to maintain desired behavior even for untested inputs. Subcriticality has the additional pragmatic benefit of mitigating physical breakdowns in logic execution (e.g., hardware bit flips) that may be of concern for some systems in addition to logic design flaws.

Formal methods and complex systems theory can also increase the power of other development practices such as fuzzing (automated randomized testing). Previous work [2] has developed an analysis to systematize the role of semantic understanding in the selection of test inputs. The essential concept is that testing is best focused on inputs that can be constructed in a simple way from information that is already known about the system – because (1) these simpler spaces are smaller and easier to cover, (2) an attacker seeking exploits will likely also start with such spaces for the same reason, and (3) evidence suggests that inadvertent design faults are more common in such spaces. Undirected, purely random inputs deserve a relatively small amount of testing, while inputs close to the nominal semantics (e.g., modifications and rearrangements of expected input) deserve strongly focused testing. As an example, weak passwords are frequent vulnerabilities that can be found by searching inputs constructed in a simple way from known information – e.g., from dictionary words.

We believe that future work can leverage these concepts to develop new programming models, seeking to reduce the dispersion of faults over the input space so that tests can become more narrowly targeted and more effective. This would draw on insights from computational complexity and machine learning. A close relationship between *testability* and *learnability* has been noted [18], [19]. Constraints that reduce information propagation and confer "smoothness" have been shown to improve learnability of digital circuit behavior. Low-depth circuits [20] that compute their output from a limited number of logic steps, and monotone circuits [21] that cannot perform logical inversion operations, both have efficient learning algorithms with bounded error. This corresponds to a greater ability for testing to provide confidence in the correctness of an implemented circuit, if the desired function can be programmed within such a constraint. It would be useful to generalize from these initial examples of improved learnability and testability to create a more broadly useful programming model for hardware and software, in which realistic applications could be implemented. There appears to be ample objective evidence [22], [23] that the programming language for an implementation makes a measurable difference in the type and quantity of vulnerabilities. This would be another example of extending some of the benefits of formal methods to systems of a scale beyond exhaustive verification.

V. Boolean Network Example

To illustrate the ideas discussed, we use Boolean networks (BNs) as a flexible representation of digital logic. BNs are effectively a simple programming language that corresponds closely to hardware sequential logic gates but also can serve as a representation of software. BNs are directed graphs in which each node has two possible states, 0 and 1. A node's state transition at each discrete time step is determined by a truth-table function of its input connections, called a transfer

function. BNs were developed as models of genetic regulatory networks and biological evolution, and have been shown to capture the essential features of more general nonlinear dynamics in complex networks [24]. BNs suit our purpose by providing a programming model with adjustable critical properties allowing the exploration of criticality effects on the propagation and damping of errors.

BNs are an example of what are known as non-uniform models of computation [25], where a digital system is finite and self-contained, takes inputs of a bounded size, and has to be made more complex in order to handle a more complex input. By contrast, conventional software and even high-level hardware design is based on a uniform model of computation, the Turing machine, which idealizes part of the digital system as unbounded storage in order to reduce the complexity of the problem-specific part of the system – the "algorithm" or "head" of the Turing machine – while handling input of arbitrary size. This idealization has been very powerful in enabling digital system development to date – yet it belies both the physical reality of finite-size computers and the common character of adaptive systems in other domains, such as biology, which are non-uniform. Formal methods, moreover, show increased tractability for bounded digital designs such as state machines, compared to general-purpose (uniform) processors and software. Results from automata theory indicate that more powerful reasoning about behavior becomes possible as models are restricted, e.g., from Turing machines to pushdown automata to finite-state machines [26]. Similarly, analyzability has been improved in practice by the use of restricted, domain-specific programming and modeling languages [27].

Though further research is needed, it is plausible that non-uniform computation is a more fruitful representation for improving both efficiency and robustness as computers become increasingly complex. As pointed out previously (Section IV), typical (uniform) computer software is found to be chaotic. With its adaptable stability properties, a BN-like model of computation may be better suited to applications where robustness is needed, such as embedded control systems. While a conventional programming model tends to exhibit instability to perturbations, a conventional Turing-complete computer can still simulate non-uniform computation and, by adopting a criticality-adjustable programming model, can benefit from its possible advantages.

As a simple example, we present here a comparison of BNs created to compute a half-adder function, adding two 1-bit numbers and producing a carry and sum bit. While it would be easy to correctly implement and exhaustively test such a simple function, we here "induce" a relatively complex implementation in a more generic fashion constrained only by the structural properties of the BN that confer its stability characteristics. This is meant to exemplify the behavior of potentially imperfect implementations of more complex requirements. A broad advantage of implementing computer systems via a formal methods/dynamical systems approach is the ability to create circuits and programs in an automated and systematic (rather than humanly idiosyncratic) manner, enabling a more objective characterization of their reliability and security.

There are many ways of composing logic gates to implement the half-adder function, and we note that in more realistic cases any actual implementation is likely to be far from "minimal". Here we randomly sample 20-node BNs, with uniformly random transfer functions, and with connectivity that is biased to propagate information from the 2 input nodes to the 2 output nodes but is otherwise random. We select those BNs that compute the correct half-adder result (for all combinations of input values) when the input nodes are clamped, the other nodes are initialized to 0, the circuit is advanced for 20 time steps, and then the output nodes are read off. This random sampling is a degenerate case of adaptive techniques such as genetic programming that could search the implementation space more efficiently.

We have generated the BNs from different ensembles that vary in the amount of connectivity, described by k, the average number of inputs per node. Though the BNs all, by construction, compute the half-adder function correctly when operating with their nominal logic, they differ in their response when bit errors are imposed. Such bit errors can represent a *breakdown* of the digital model, as for physical upsets in hardware, or alternatively the effect of untested states or unanticipated input *within* the digital space of a complex design.

In general, the response of a dynamical system to perturbations is described by a "Lyapunov exponent" measuring how perturbations are damped or amplified by the dynamics. Damping indicates a subcritical, stable, quiescent system; amplification indicates a supercritical, unstable, chaotic system; and the borderline is the "edge of chaos" mentioned in Section IV. Mathematical work on generic BN ensembles has shown how the connectivity and transfer functions control the transition from quiescent to chaotic behavior [17], [28]. Here we show evidence that similar relations hold for BNs programmed (via selection) to perform a particular task.

Figure 1 shows the structure and final output state of two typical half-adder BNs, one from the ensemble with $k = 1.5$ and one from the ensemble with $k = 2.5$, with a 1% bit error rate per node update in both cases. The correct output for the inputs shown $(1 + 1)$ would be a carry of 1 and a sum of 0. All nodes that differ at the last time step from their states in an error-free run are outlined in red. In this case, the $k = 1.5$ network shows damping of perturbations and gets the correct answer, while the $k = 2.5$ network shows amplification of perturbations and gets the wrong answer.

As indicated, over the ensemble of such BNs, the $k = 1.5$ circuits have a significantly lower expected error in the final output than the $k = 2.5$ circuits. This is in accordance with the simplest dynamical systems characterization of purely random BNs, which show quiescence for $k < 2$ and chaos for $k > 2$ [17]. Indeed, as shown by the ensemble simulation results in Figure 2, the parameter k describes a systematic relation between the local occurrence of unexpected erroneous states and the resulting loss of system-level correctness. Requiring subcriticality is a constraint that makes generally makes a

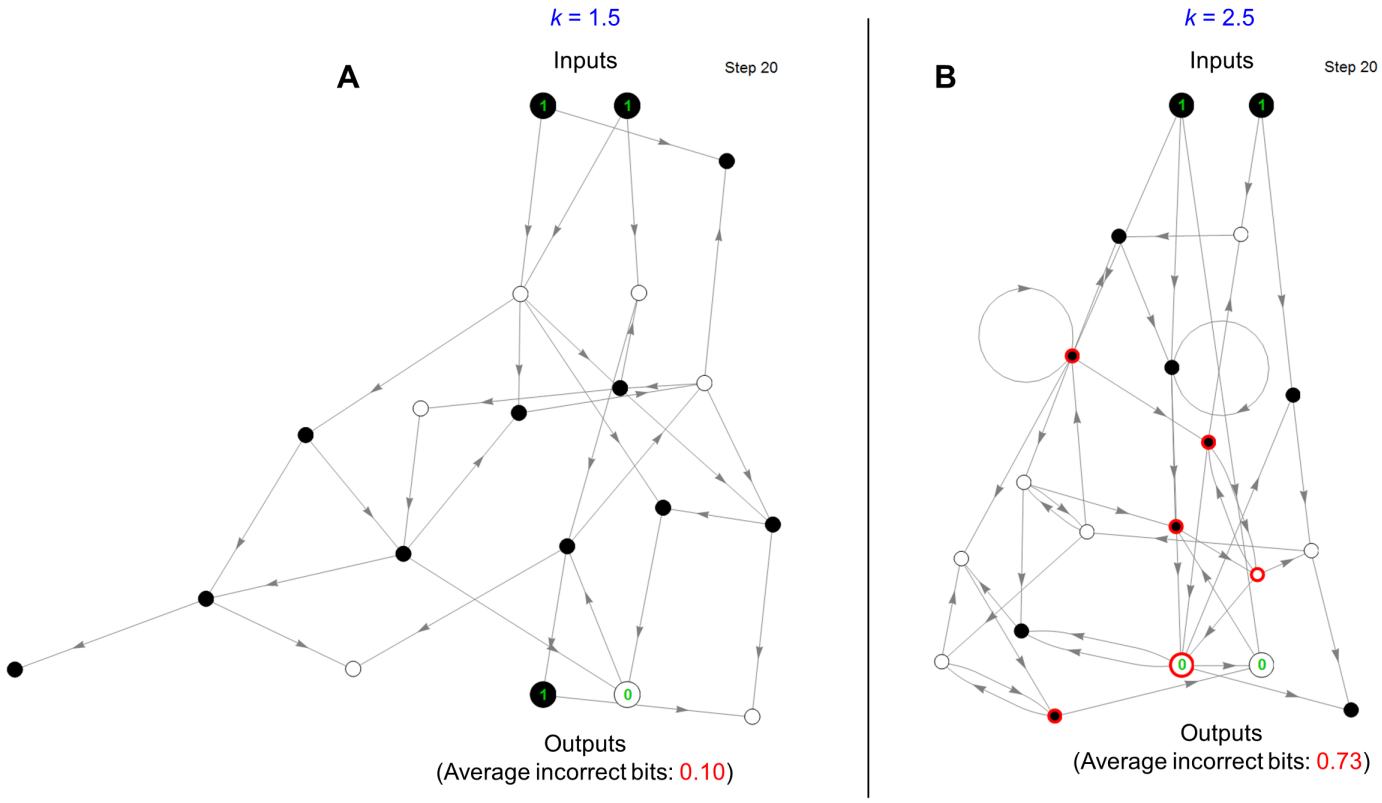

Fig. 1. Final state of two Boolean networks that nominally compute the half-adder function, each drawn from a different ensemble based on k, the average number of inputs per node ($k = 1.5$ is quiescent and $k = 2.5$ is chaotic). A 1% bit error rate per node update is imposed. All nodes that differ at the last time step from their states in an error-free run are outlined in red.

digital design more difficult to create but confers valuable predictability on behavior – aspects shared with the more specific approach of formal methods. Design parameters such as k enable probabilistically assessing potential catastrophic failures too rare to be found through testing, and remain relevant even when the full system scale is beyond the reach of exhaustive verification.

The present example is, however, simple enough to illustrate formal methods as a base case for assessing and enhancing robustness. We have analyzed the two example BNs in Figure 1 using the NuSMV model checker [29]. Our initial formal analysis allows the possibility of any single bit error during some range of time steps, via a nondeterministic model similar to one previously used for upsets in hardware [30]. We seek to exhaustively prove or disprove correct function of the half-adder circuits under this error model. For instance, the correctness requirement for the carry bit is represented symbolically in NuSMV as

```
LTLSPEC F ((clock=20) & (n18 = (n00&n01)))
```

which asserts that once the circuit has advanced through 20 time steps, the carry bit `n18` equals the logical AND of the two input bits `n00` and `n01`.

NuSMV finds that the output of the chaotic circuit is susceptible to corruption from a bit error occurring at *any* time step of the run. By contrast, NuSMV proves that the output

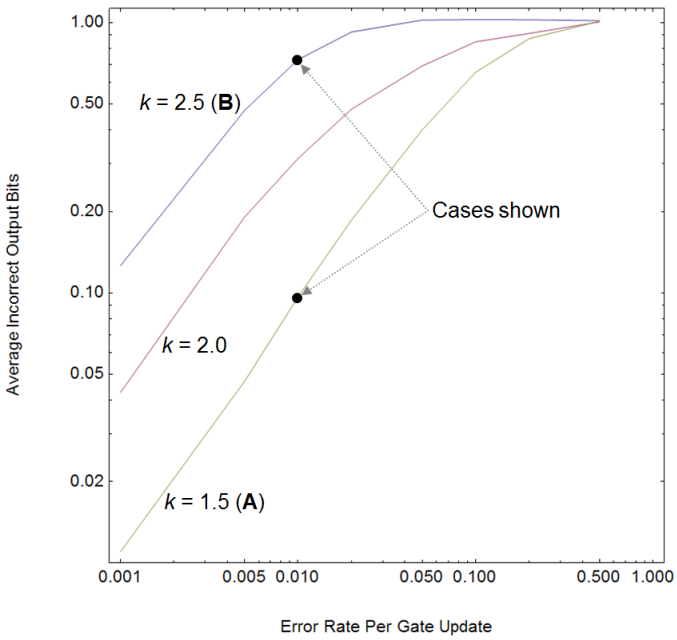

Fig. 2. Simulations of half-adder Boolean networks: Average number of output bits that are incorrect as a function of the bit error rate, for different values of k, the average number of inputs per node. The points correspond to the Boolean networks shown in Figure 1. Note that when the bit error rate is 0.5, all logic gates (including the outputs) act as uniform random number generators, so on average 1 out of the 2 output bits is incorrect.

of the quiescent circuit can be corrupted *only* if the bit error occurs in the last 5 of the 20 time steps, and is self-healing otherwise. Thus, the exhaustive formal verification results for these example circuits confirm the insights from dynamical systems theory.

VI. Conclusion

Evidence has been provided that design of digital systems from a formal model yields a qualitatively more robust result than would be expected otherwise – in many cases for fairly complex applications. A case is made that the measure of this robustness is well beyond that which could be expected as a result of satisfying the proof obligations alone, but rather that a broader principle is at work. We argue that formal modeling languages and tools along with imposed proof obligations constrain the program design process in ways that, by themselves, increase the robustness of the resulting implementation.

We offer an analogy through programming Boolean networks where these principles can be explored more directly and quantitatively. By selecting the network to be subcritical, the resulting implementation becomes more robust to faults and vulnerabilities, but also becomes harder to program. However, as the network becomes critical, and then supercritical, it is easier to program but also considerably less robust. Subcritical, overdamped Boolean networks are easier to analyze in the sense that they are more predictable. Supercritical, chaotic Boolean networks are harder to analyze in the sense that they are random. In analogy, we suggest that the more constraining formally informed design process, which is more concerned with transparency to analysis and less with programmability, yields a more robust result for similar reasons. More data regarding real-world results of formally informed design of complex digital systems will likely be available as time goes on and will help confirm or contradict this hypothesis, but for the present this analogy seems warranted.

Acknowledgment

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under contract DE-AC04-94AL85000.

References

[1] R. C. Armstrong and J. R. Mayo, "Leveraging complexity in software for cybersecurity," in *Proc. 5th Cyber Security and Information Intelligence Research Workshop*, Oak Ridge, TN, Apr. 2009.

[2] J. R. Mayo and R. C. Armstrong, "Tradeoffs in targeted fuzzing of cyber systems by defenders and attackers," in *Proc. 7th Cyber Security and Information Intelligence Research Workshop*, Oak Ridge, TN, Oct. 2011.

[3] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM Computing Surveys*, vol. 41, p. 19, 2009.

[4] J. Leyden, "Patch bash now: 'shellshock' bug blasts OS X, Linux systems wide open," Sep. 2014. [Online]. Available: http://www.theregister.co.uk/2014/09/24/bash_shell_vuln/

[5] D. Peled, "Ten years of partial order reduction," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, A. Hu and M. Vardi, Eds. Springer Berlin Heidelberg, 1998, vol. 1427, pp. 17–28.

[6] "List of model checkers." [Online]. Available: http://en.wikipedia.org/wiki/List_of_model_checking_tools

[7] P. C. Hickey, L. Pike, T. Elliott, J. Bielman, and J. Launchbury, "Building embedded systems with embedded DSLs (experience report)," in *Proc. International Conference on Functional Programming (ICFP)*, Gothenburg, Sweden, Sep. 2014. [Online]. Available: http://www.cs.indiana.edu/~lepike/pub_pages/icfp14.html

[8] P. Paganini, "Hack-proof drones possible with HACMS technology," Jun. 2014. [Online]. Available: http://resources.infosecinstitute.com/hack-proof-drones-possible-hacms-technology/

[9] L. Kelion, "Parrot drones 'vulnerable to flying hack attack'," Dec. 2013. [Online]. Available: http://www.bbc.com/news/technology-25217378

[10] X. Leroy, "Formal verification of a realistic compiler," *Communications of the ACM*, vol. 52, no. 7, pp. 107–115, 2009.

[11] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and understanding bugs in C compilers," *SIGPLAN Notices*, vol. 46, no. 6, pp. 283–294, Jun. 2011.

[12] J. Zhao, S. Nagarakatte, M. M. Martin, and S. Zdancewic, "Formalizing the LLVM intermediate representation for verified program transformations," *SIGPLAN Notices*, vol. 47, no. 1, pp. 427–440, Jan. 2012.

[13] J. Rushby, "Formal methods and the certification of critical systems," Computer Science Laboratory, SRI International, Menlo Park, CA, Tech. Rep. SRI-CSL-93-7, Dec. 1993, also issued under the title *Formal Methods and Digital Systems Validation for Airborne Systems* as NASA Contractor Report 4551, December 1993.

[14] C. Newcombe, T. Ratha, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "Use of formal methods at Amazon Web Services," Sep. 2014. [Online]. Available: http://research.microsoft.com/en-us/um/people/lamport/tla/amazon.html

[15] T. Mytkowicz, A. Diwan, and E. Bradley, "Computer systems are dynamical systems," *Chaos*, vol. 19, p. 033124, 2009.

[16] J. M. Carlson and J. Doyle, "Highly optimized tolerance: A mechanism for power laws in designed systems," *Physical Review E*, vol. 60, pp. 1412–1427, 1999.

[17] S. A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[18] H. Zhu, P. Hall, and J. May, "Inductive inference and software testing," *Software Testing, Verification and Reliability*, vol. 2, pp. 69–81, 1992.

[19] K. Romanik, "Approximate testing and its relationship to learning," *Theoretical Computer Science*, vol. 188, pp. 79–99, 1997.

[20] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, Fourier transform, and learnability," *Journal of the ACM*, vol. 40, pp. 607–620, 1993.

[21] N. H. Bshouty and C. Tamon, "On the Fourier spectrum of monotone functions," *Journal of the ACM*, vol. 43, pp. 747–770, 1996.

[22] WhiteHat Security, "2014 website security statistics report," Apr. 2014. [Online]. Available: http://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf

[23] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in Github," in *Proc. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong, China, Nov. 2014, pp. 155–165.

[24] L. Glass and S. A. Kauffman, "Logical analysis of continuous, nonlinear biochemical control networks," *Journal of Theoretical Biology*, vol. 39, pp. 103–129, 1973.

[25] O. Goldreich, *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[26] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Pearson, 2014.

[27] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, Dec. 2005.

[28] C. Seshadhri, Y. Vorobeychik, J. R. Mayo, R. C. Armstrong, and J. R. Ruthruff, "Influence and dynamic behavior in random Boolean networks," *Physical Review Letters*, vol. 107, p. 108701, 2011.

[29] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Proc. 14th International Conference on Computer Aided Verification*, Copenhagen, Denmark, Jul. 2002, pp. 359–364.

[30] S. A. Seshia, W. Li, and S. Mitra, "Verification-guided soft error resilience," in *Proc. Conference on Design, Automation and Test in Europe*, Nice, France, Apr. 2007, pp. 1442–1447.