Leveraging Abstraction to Establish Out-of-Nominal Safety Properties

Jackson R. Mayo(✉), Robert C. Armstrong, and Geoffrey C. Hulette

Sandia National Laboratories, P.O. Box 969, Livermore, CA 94551-0969, USA
{jmayo,rob,ghulett}@sandia.gov

Abstract. Digital systems in an out-of-nominal environment (e.g., one causing hardware bit flips) may not be expected to function correctly in all respects but may be required to fail safely. We present an approach for understanding and verifying a system's out-of-nominal behavior as an abstraction of nominal behavior that preserves designated critical safety requirements. Because abstraction and refinement are already widely used for improved tractability in formal design and proof techniques, this additional way of viewing an abstraction can potentially verify a system's out-of-nominal safety with little additional work. We illustrate the approach with a simple model of a turnstile controller with possible logic faults (formalized in the temporal logic of actions and NuSMV), noting how design choices can be guided by the desired out-of-nominal abstraction. Principles of robustness in complex systems (specifically, Boolean networks) are found to be compatible with the formal abstraction approach. This work indicates a direction for broader use of formal methods in safety-critical systems.

Keywords: Abstraction · Refinement · Model checking · Fault tolerance · Soft errors · Temporal logic of actions · NuSMV

1 Introduction

Due to the combinatorial complexity of digital systems, not only is exhaustive testing infeasible as a means to ensure safety, but even the reasoning techniques used by formal methods face scalability challenges in verifying large designs and complex safety requirements. A widely used technique to improve the tractability of formal verification is to work with abstractions (or overapproximations), which can be simpler to analyze and are conservative in the sense that their verified safety properties are guaranteed to hold also in the actual implementation. This guarantee applies because a valid abstraction permits all behaviors that occur in the implementation and possibly additional behaviors. In current formal methods, abstractions are used in two main contexts:

1. *Proof techniques* that search for a post-hoc abstraction suitable for verifying desired properties of a given implementation, as in counterexample-guided abstraction refinement (CEGAR) [3].

C. Artho and P.C. Ölveczky (Eds.): FTSCS 2015, CCIS 596, pp. 172–186, 2016.
DOI: 10.1007/978-3-319-29510-7_10

2. *Design techniques* that start from an abstraction in which desired properties can be proven and then create an implementation by refinement, as in the Event-B method [1].

In both cases, the abstraction is a means to an end: either generating a proof of an existing design, or generating a provable design. The abstraction is of value because it can be tractably verified for safety and because it has an overapproximation relationship to the implementation, but serves little purpose beyond these points. If the implementation could be verified directly, the need for the abstraction would be obviated.

Here we present a different perspective on abstraction – useful when, under some conditions, a system is physically capable of additional behaviors beyond its "nominal" operation. In this approach, we note that a typical formal model of the implementation makes certain assumptions about the environment that are not universally valid. Thus, the requirements that are verified on this implementation model, which may include not only safety but also reliability, etc., are proven to hold in this nominal environment. This is practically sufficient for some requirements, given that the nominal environment can be maintained often enough for the system to be useful. But critical safety properties may need to be guaranteed under a less restrictive model that permits particular "out-of-nominal" behaviors, if such behaviors may physically occur often enough to be of concern for the risk of catastrophic failure. Our observation is that the abstraction concept, already commonly used in formal methods as a mathematical technique, can be reinterpreted as defining a space of possible "real-world" out-of-nominal behaviors for which the abstraction-verified safety properties are still guaranteed to hold. Thus, by leveraging suitable abstractions, we can gain out-of-nominal safety verification for free.

A primary example of out-of-nominal behavior is the response of digital hardware to electrical or other physical stimuli that produce states not accounted for in the logic design – with the abnormal physical dynamics generating a nominally disallowed digital state transition such as a bit flip. A variety of formal techniques have been investigated for modeling and verifying such behavior [4,7,8]; recognizing that out-of-nominal behavior may overlap with other formal abstractions can increase the applicability of these techniques, particularly in earlier stages of the design process. More generally, other types of unexpected but not totally unforeseeable inputs from the environment can be treated as out-of-nominal behavior. For example, in modular verification of a system where each component is verified subject to assumptions on the behavior of other components with which it interacts, a conservative approach that verifies safety for a suitable overapproximation can create a "firebreak" around each component that mitigates the possibility of catastrophic cascading failure in the event of isolated malfunctions. A complex systems theory of such firebreaks has been developed previously [15].

In the remainder of this paper we present the formal abstraction framework for understanding out-of-nominal behavior (Sect. 2), the definition of a simple example model of a turnstile (Sect. 3), an illustration of the framework using the

example (Sect. 4), a conclusion (Sect. 5), and the formalization of aspects of the example in the temporal logic of actions or TLA (Appendix A) and in NuSMV (Appendix B).

2 Modeling Out-of-Nominal Safety Properties

The safety properties of a given model are required to hold at all times over all possible behavioral paths. Such properties, when imposed on an abstraction, require that every path in the abstraction conforms to the properties, and thus every refinement will as well. The use of abstraction in verifying safety requirements is well established.

Here we distinguish "critical" safety requirements that must hold even in out-of-nominal environments (Fig. 1). These out-of-nominal fail-safe requirements are less strict (allow more behaviors) than the requirements for nominal operation and thus constitute an abstraction of the nominal requirements. Safety-critical devices where failure modes can be anticipated are likely candidates for this technique. Nominal requirements can be relaxed to admit acceptable modes of failure. The resulting out-of-nominal safety requirements reflect an engineering decision that certain properties must be preserved even in exceptional circumstances that may be considered unlikely to occur.

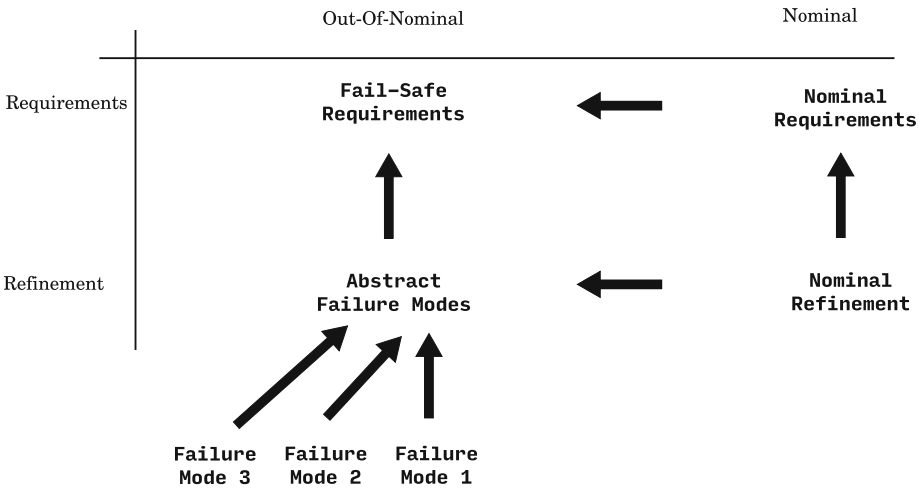

Fig. 1. Refinement/abstraction conceptual diagram for treating out-of-nominal and nominal models in a unified way. The arrows point in the direction of abstraction.

The safety requirements must ultimately be verified on formal models that reflect the actual nominal and out-of-nominal behavior of the system being designed. Such models are typically tied to the requirements via one or more

abstraction/refinement steps ultimately leading to a model of a practical implementation. In our approach, upon refinement, the out-of-nominal model remains an abstraction of the nominal one (Fig. 1). By stipulating that the out-of-nominal refinement has a superset of the behaviors of the nominal refinement, we ensure that the safety properties verified for out-of-nominal operation also hold for nominal operation. These critical safety properties take the form of a fail-safe mode where nominal function is no longer guaranteed but essential safety invariants still hold. Of course, the approach is limited to those out-of-nominal failure modes that can be foreseen and modeled.

Not all foreseeable failure modes may manifest an abstraction or overapproximation of the system's nominal behavior. A particular failure mode may render the system incapable of performing some nominal behaviors. The removal of possible behavioral paths, by itself, does not invalidate any of the nominal safety properties, but can affect functional requirements that are outside the scope of the formal refinement methodology applied in this work. Out-of-nominal scenarios of concern for safety would involve adding at least some new behaviors. In typical cases, failures can occur to varying degrees or not at all depending on practically unpredictable events. Thus, it is often natural for out-of-nominal behavior to be represented in a way that includes nominal behavior as a possibility. Regardless, an out-of-nominal model can be *made* an overapproximation by simply adding the nominal behavior to it as an allowed nondeterministic branch.

If we are to apply critical safety requirements globally across all failure modes, then the high-level out-of-nominal refinement will represent the union of all failure modes together with the nominal behavior. In this way, all models of particular failure modes are refinements of the global failure refinement and inherit any safety property proven for this global refinement. The nominal model is *also* a refinement and inherits the same safety properties (Fig. 1). Not admitted in this work is a case where a safety property is required to hold *only* for out-of-nominal operation and is not present in the nominal model. Though such cases exist, it is considered rare for a nominal implementation to lack a safety requirement present in a failure mode for that system.

Viewing behaviors of anticipated malfunctions as an abstraction of the nominal behavior has some advantages. For complex safety-critical systems that are prone to failure, it is important to "design-in" anticipated failures with their own fail-safe requirements. Recasting such requirements into the familiar abstraction/refinement design practice means that the same tools can be brought to bear on these designed-in benign failure requirements as part of the normal design process. Another advantage is that anticipated failure modes are incorporated into the design process up-front rather than as an afterthought.

3 Example Turnstile Model

For an illustration, we use the familiar turnstile model [6] in simplified form. A turnstile requires a coin to permit the patron admission by pushing on the bar. In a simplified description, we can identify three Boolean state variables

for the device: C, P, and L, indicating whether a coin is present, whether the bar is being pushed, and whether the bar is locked. We idealize the operation of the turnstile as a sequence of discrete instants at which C and P can be set arbitrarily from the outside and L updates at the next instant in response. If the coin is present and the bar is locked, the bar should become unlocked and remain so until the patron pushes through, after which it should become locked again. If the coin is absent, the bar should remain locked. We can synthesize the desired nominal properties into a TLA+ [11] formula:

$$\begin{array}{lll} S1 & \triangleq (\neg C \wedge L \Rightarrow L') \ldots\ldots & \textbf{critical safety property} \\ S2 & \triangleq (C \wedge L \Rightarrow \neg L') & \\ S3 & \triangleq (\neg P \wedge \neg L \Rightarrow \neg L') & (1) \\ S4 & \triangleq (P \wedge \neg L \Rightarrow L') & \\ \mathit{Safety} & \triangleq \Box[S1 \wedge S2 \wedge S3 \wedge S4]_{\langle C,P,L\rangle}. & \end{array}$$

Here, each Sn defines a safety property in terms of a TLA action, which relates the variables C, P, and L in the "current" instant to L', representing the value of L in the "next" instant. TLA formulas describe *behaviors*, infinite sequences of states over a set of named variables, and so we have to lift the description of individual steps into a predicate on behaviors. To combine the safety properties into the requirement *Safety*, we require that each step must satisfy the conjunction of the safety properties, or else be a "skip" step where the next state is identical to the current one. In TLA+ this is expressed as $\Box[S1 \wedge S2 \wedge S3 \wedge S4]_{\langle C,P,L\rangle}$.

While all of the implications in (1) can be thought of as safety properties, the "critical safety property" $S1$ is one that we wish to preserve in a design for anticipated out-of-nominal conditions. We could have designated another one (or more) of the safety conditions as "critical" – there is nothing special about the property $S1$ other than our choice of it for this example. We can interpret $S1$ as "the turnstile will remain locked unless a coin is present" $(\neg C \wedge L \Rightarrow L')$. Out-of-nominal designs will be discussed further in Sect. 4.

The nominal requirements in (1) can be used as an abstraction suitable for refinement. If the refinement is valid, all of $S1$ through $S4$ will be true of the implementation. One initial refinement of the requirements is described by the action

$$L' = (\neg C \wedge L) \vee (P \wedge \neg L), \tag{2}$$

and this can be elaborated into a full TLA+ model, shown in Fig. 4 in Appendix A. The TLC model checker can prove that the behaviors of this model, encoded in a TLA formula *Spec*, refine *Safety*, i.e., satisfy the safety conditions $S1$ through $S4$. Since the model is finite, TLC readily verifies that $\mathit{Spec} \Rightarrow \mathit{Safety}$.

The refinement (2) would need to be "compiled" (i.e., further refined) into a program running on a processor, or in the ensuing example for this paper, synthesized into logic gates. It is the specifics of the implementation that determine whether this circuit is robust to the anticipated failure modes.

4 Design and Out-of-Nominal Verification via Abstraction

4.1 Refinement (High Level)

We now consider a method by which abstraction and refinement can be used in a formal design process in order to account for out-of-nominal conditions. The process starts, as any design process should, with the requirements. These are gathered in the usual ways and must be formalized. These are the nominal requirements.

Next, certain of these requirements are designated as "critical" – these are the out-of-nominal requirements, i.e., those that must hold even under some (predicted) mode of system failure or inconsistency. Our methodology dictates that now the designer must prove that the nominal requirements refine the out-of-nominal requirements. If the out-of-nominal requirements are a subset of the nominal requirements then this proof is trivial, since any system behavior satisfying a set of requirements will also satisfy any subset of those requirements.

Next, we refine the nominal requirements. The refined model is closer to an implementation, although it may still be quite abstract. Refinement of the nominal model is done in the usual way [1,11], ensuring that the level above simulates the level below.

Finally, we must construct the out-of-nominal refinement such that it both *refines* the out-of-nominal requirements *and abstracts* the nominal refinement, completing the commuting square diagram (shown for the turnstile example in Fig. 2). This step might be quite difficult, and we know of no general approach to construct this model. However the turnstile example may be typical of certain cases. In this case, our out-of-nominal requirement is only that $\neg C \wedge L \Rightarrow L'$. In the nominal refinement, L evolves based on the action

$$L' = (\neg C \wedge L) \vee (P \wedge \neg L).$$

Since the first disjunct alone already satisfies the out-of-nominal requirement that $\neg C \wedge L \Rightarrow L'$, we can consider the second disjunct to behave "randomly" and, at any step, draw its value from either the nominal behavior $P \wedge \neg L$ or its negation $\neg(P \wedge \neg L)$. In the model, we denote by X a value from this set, and the out-of-nominal refinement is derived by replacing the action above with

$$L' = (\neg C \wedge L) \vee X.$$

This model is shown in Fig. 5 in Appendix A. We have verified with TLC that it both refines the out-of-nominal requirements and abstracts the nominal refinement, thus completing the commuting diagram.

By contrast, if we had used the logically equivalent nominal refinement

$$L' = (\neg C \vee \neg L) \wedge (P \vee L),$$

it would not have been straightforward to obtain an out-of-nominal abstraction preserving the critical safety requirement $S1$. That is, while the disjunctive and

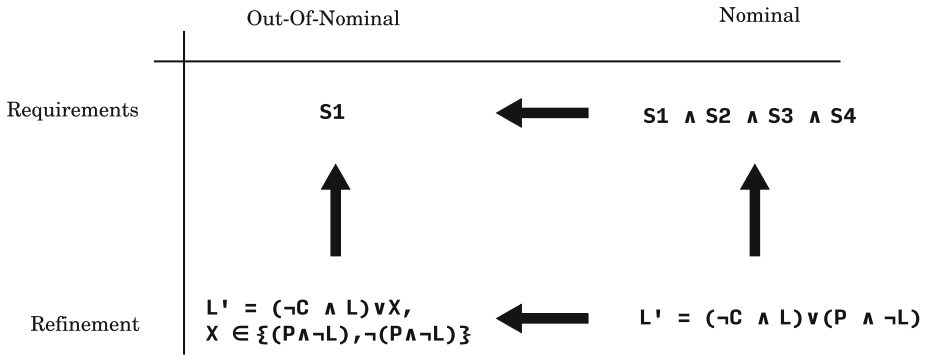

Fig. 2. Refinement/abstraction diagram for the turnstile example. The arrows point in the direction of abstraction. Existing formal abstractions can be reinterpreted in this framework; a technique like CEGAR might already prove that the nominal design (lower right) satisfies a safety property (upper left) by finding an abstraction (lower left) that satisfies the safety property.

conjunctive normal forms are of course equivalent in their nominal behavior, in this example one particular choice of design offers the ability to tolerate a faulty out-of-nominal operation. This interpretation gives abstraction an even more central role in driving the design process.

It is useful to ask: How generalizable and automatable is the use of abstraction techniques to understand out-of-nominal behavior? While we present only a preliminary exploration of this type of approach, we suggest that there are likely insights to be gained on many specific digital system models by viewing already-used abstraction techniques through the out-of-nominal lens. In traditional nominal verification, discovering a useful abstraction in which given safety properties can be proven is typically an iterative process, either automated or manual. The goal of capturing some realistic out-of-nominal behavior in the abstraction can be an additional criterion guiding this process.

For example, in design by refinement, a high-level model satisfying critical safety properties could be constrained to be assembled from abstracted component models that are known to represent the behavior of implementable devices including both nominal and out-of-nominal environments of interest. This would ensure that subsequent refinement can match a physically realizable implementation while preserving the out-of-nominal requirements. Moreover, the choice of physical implementation itself could be directly informed by abstractions that are found in other ways. If CEGAR is applied to a critical safety property and discovers a suitable abstraction of the nominal model automatically, the system design could be adjusted to ensure that its out-of-nominal behavior falls within this abstraction. In the turnstile model, CEGAR might produce the abstraction $L' = (\neg C \wedge L) \vee X$ in the course of proving $\neg C \wedge L \Rightarrow L'$. More realistic applications of CEGAR [10] result in other abstractions that may correspond to out-of-nominal behavior, such as allowing the values of variables to be corrupted

as long as certain predicates are not altered. This could define the strength of error correction needed in an implementation.

4.2 Implementation (Low Level)

We now discuss how the refined logic design for the turnstile (on both the out-of-nominal and nominal sides) can be related to a notional implementation in hardware gates. This corresponds to adding another level of detail to the model that could be reified in raw gates, moving from the second to the third row in Fig. 3. We could initially interpret the nominal logic $L' = (\neg C \wedge L) \vee (P \wedge \neg L)$ directly in terms of AND and OR gates. Then the out-of-nominal logic $L' = (\neg C \wedge L) \vee X$ implies that the $P \wedge \neg L$ term can be computed by an unreliable gate, but the remaining gates must remain reliable even under out-of-nominal conditions. Often this is achieved using some physically more robust but more expensive type of gate, and is ineluctably tied to the physical failure mode(s) that the designer has in mind. To illustrate an alternative technique, we discuss an intrinsically robust implementation using Boolean networks (BNs) informed by principles of digital error damping. Such BNs have several advantages:

1. The analysis draws on the rich body of science developed for BNs [9] as previously applied to discrete system robustness, including digital and biological applications; error creation, propagation, and extinction in BNs are well characterized.
2. The statistics of error damping in BNs have been previously evaluated [12] for a digital half-adder. Because of this, the example implementations used here are known to be representative of the class of BNs from which they are chosen.
3. The dynamical systems principles illustrated by BNs are applicable to much more complex designs than the turnstile example and to broader types of faults, offering a means of assessment even for systems beyond the reach of exhaustive formal verification.

We draw on previous work [12] in which example BNs were constructed to compute a half-adder function and their robustness was analyzed with the NuSMV [2] model checker. For present purposes, we ignore the "sum" output and use only the "carry" output, which corresponds directly to an AND operation. Conventionally, a BN is interpreted as a sequential logic circuit. To implement combinational logic, we replicate the gates in "tiers", with each tier providing its results as input to the next, and with the final output being read at the end of a specified number of tiers (here, 20). This corresponds to "unrolling" the conventional BN steps and can analyzed identically using model checkers, etc. The BNs are used here as a notional means of implementing the turnstile's combinational logic in a way that is systematic (rather than idiosyncratic) and representative of more complex designs.

Two BNs were constructed, differing in the design parameter k, the average number of inputs per node [12]. In accordance with complex systems analysis [9],

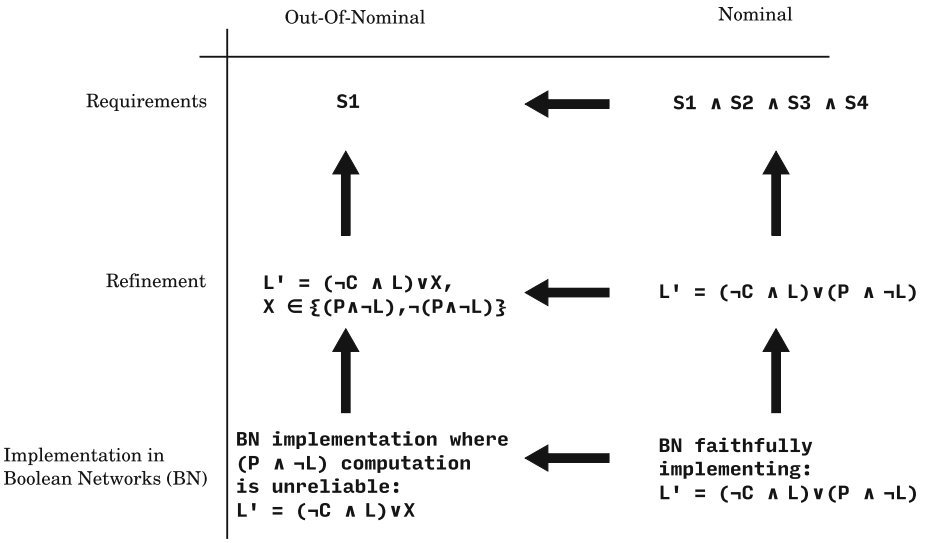

Fig. 3. Continuation of Fig. 2 where we add an implementation in gate-level Boolean networks. It is at this lowest implementation level that the failure mode will evidence itself and must be anticipated and accounted for in the out-of-nominal design.

the BN with $k = 1.5$ shows "quiescent" behavior (perturbations are damped) and the BN with $k = 2.5$ shows "chaotic" behavior (perturbations are amplified). Typical real-world digital implementations are found empirically to be chaotic [13]; such implementations are cheaper to create because they impose fewer restrictions on programmability. Quiescent implementations that damp bit-flip errors are more constrained and generally more difficult to create. Our strategy here is to use the cheaper chaotic implementation for parts of the design that do not impact the critical safety property, and to use the more expensive quiescent implementation for parts that need robustness to preserve the critical safety property.

In using the BNs for the turnstile, we take advantage of the higher-level abstraction properties already established. Specifically, we implement each of the two AND operations in $L' = (\neg C \wedge L) \vee (P \wedge \neg L)$ with a BN. This means that the two values $\neg C$ and L are wired to the inputs of a BN and the carry output is used for the result $\neg C \wedge L$, and similarly for $P \wedge \neg L$. We assume that the other operations, such as the NOT initially applied to some inputs and the OR performed at the end, are fully reliable for this example.

Each of the two AND operations in $L' = (\neg C \wedge L) \vee (P \wedge \neg L)$ can be implemented with either of the BNs as far as *nominal* behavior is concerned. This is verified by exhaustive testing as well as model checking with NuSMV [12], and is as expected because the BNs were chosen to compute their function correctly when operating with their nominal logic. Thus, the abstraction arrow leading upward from the bottom right of Fig. 3 is valid.

For out-of-nominal behavior, as before [12], we consider the possibility of any single bit flip (incorrect gate output) within some range of tiers in the BN, again using a nondeterministic formal model of the kind used in other work on soft errors [14]. We have adapted the NuSMV analysis in this case to check the correctness of the carry bit specifically. A portion of our NuSMV model is shown in Fig. 6 in Appendix B. In these BNs, because bit flips occurring at or shortly before the output stage may not have a chance to self-correct, the bit flip is restricted to the first $n_{\max}$ tiers, where we consider $1 \leq n_{\max} \leq 20$. The NuSMV analysis finds that for no such value of $n_{\max}$ does the chaotic BN reliably implement the AND operation, while the quiescent BN does so for any $n_{\max} \leq 15$. That is, most of the computations performed by the quiescent BN can be susceptible to a bit flip, and relatively few of them (the last 5 tiers) need to be protected. Thus, if we can arrange that the effect of the out-of-nominal environment is not felt in the last 5 tiers, then the quiescent BN can be used to implement the "critical" term $\neg C \wedge L$ and correctly refines it on the out-of-nominal side – the abstraction arrow leading upward from the lower left in Fig. 3. Meanwhile, either BN (or for that matter, any nominally correct implementation) can be used for $P \wedge \neg L$ because the out-of-nominal side imposes no constraint on this term.

Hence, we have shown that for a suitable out-of-nominal environment, a BN-based implementation of the turnstile logic with quiescent $\neg C \wedge L$ and arbitrary $P \wedge \neg L$ can complete the bottom row in Fig. 3, conforming to the previous abstractions on both the out-of-nominal and nominal sides. As mentioned, quiescent implementations are harder to design, and so limiting the need for them (here to one half of the turnstile logic) is useful.

In accordance with the remarks at the end of Sect. 4.1, the relation between the higher-level models and the BN implementations illustrates the potential for two-way interaction in the design process. The robustness that is designed-in at the gate level can be targeted at the goal of making the out-of-nominal behavior conform to a chosen abstraction; resources need not be spent on correcting errors that are allowed by the abstraction. Conversely, the availability and efficiency of robust implementations can motivate the use of particular abstractions in a formal design methodology.

5 Conclusion

We have presented an approach for modeling out-of-nominal behavior in digital systems so that critical safety properties can be established, in a way that leverages existing formal design and verification techniques. Our approach takes advantage of a key observation: The relation between nominal and out-of-nominal behavior can be viewed as an instance of the same kind of formal abstraction that is used for other purposes, and so analysis techniques and specific abstractions can be shared. Nominal and out-of-nominal requirements and implementations are connected by an interlocking set of abstraction relationships.

This work can contribute to new digital design and verification techniques that ensure safety in out-of-nominal environments as an inherent property rather than addressing it after the fact. This will likely benefit from an iterative design process in which the nominal and out-of-nominal requirements and implementations can be adjusted until the network of abstractions is complete and consistent. For a design already created with only nominal analysis, abstractions can reveal what properties are preserved in what out-of-nominal environments, and thus may enlarge the usefulness of the design or suggest ways of improving it.

Possible extensions of this work include:

1. Generalizing the dichotomy of nominal and out-of-nominal to a larger family of different environments, each of which may have its own set of safety requirements based on likelihood of occurrence and consequences of failure.
2. Enabling statistical reasoning with probabilistic (rather than merely nondeterministic) models of out-of-nominal behavior, probabilistic safety requirements, and probabilistic model checkers [5], using suitable notions of abstraction and refinement.
3. Further integrating robust-design principles from formal methods and complex systems theory to enable out-of-nominal verification with as much confidence as possible for systems beyond the reach of exhaustive analysis.

Acknowledgments. 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. This work was funded by NNSA's Advanced Simulation and Computing (ASC) Program.

A High-Level Model for Turnstile in TLA

As discussed in Sect. 3 and Sect. 4.1, TLA+ is used to specify and verify both nominal and out-of-nominal models for the turnstile example. The nominal model is shown in Fig. 4 and the out-of-nominal model in Fig. 5. Both models have three variables *lock*, *coin*, and *push*, corresponding to the variables L, C, and P described in Sect. 3. The specifications are given in the idiomatic TLA+ style: *Init* constrains the initial conditions, *Next* describes the "next step" relation, and *Spec* expresses the complete temporal logic specification [11].

The relation *Next* is defined by existential quantification over parameters c and p, representing new values of *coin* and *push* in the relation *Step*. This somewhat contorted idiom is used because a step must completely describe the evolution of each variable. The existential expresses that *coin* and *push* may each evolve nondeterministically at each step.

The property *TypeInvariant* states that each variable is limited to Boolean values, while *Safety* expresses the set of safety properties drawn from $S1$ through $S4$ that apply to each model. In the nominal model, *OutOfNominalSpec* imports the out-of-nominal specification for use in proving refinement (see Sect. 4.1). The type invariant, safety, and refinement properties were checked for correctness using the TLC model checker.

MODULE *Turnstile*

VARIABLES $lock$, $coin$, $push$

$Init \triangleq$
$\quad \land lock = \text{TRUE}$
$\quad \land coin \in \text{BOOLEAN}$
$\quad \land push \in \text{BOOLEAN}$

$Step(c, p) \triangleq$
$\quad \land lock' = ((\neg coin \land lock) \lor (push \land \neg lock))$
$\quad \land coin' = c$
$\quad \land push' = p$

$Next \triangleq \exists\, c, p \in \text{BOOLEAN} : Step(c, p)$

$vars \triangleq \langle lock, coin, push \rangle$

$Spec \triangleq Init \land \Box[Next]_{vars}$

$TypeInvariant \triangleq$
$\quad \land lock \in \text{BOOLEAN}$
$\quad \land coin \in \text{BOOLEAN}$
$\quad \land push \in \text{BOOLEAN}$

THEOREM $Spec \Rightarrow \Box\, TypeInvariant$

$S1 \triangleq \neg coin \land lock \Rightarrow lock'$
$S2 \triangleq coin \land lock \Rightarrow \neg lock'$
$S3 \triangleq push \land \neg lock \Rightarrow lock'$
$S4 \triangleq \neg push \land \neg lock \Rightarrow \neg lock'$

$Safety \triangleq \Box[S1 \land S2 \land S3 \land S4]_{vars}$

THEOREM $Spec \Rightarrow Safety$

$OutOfNominal \triangleq$ INSTANCE $OutOfNominal$
$OutOfNominalSpec \triangleq OutOfNominal!Spec$

THEOREM $Spec \Rightarrow OutOfNominalSpec$

Fig. 4. TLA+ specification for the nominal turnstile.

MODULE *OutOfNominal*

VARIABLES $lock$, $coin$, $push$

$$
\begin{aligned}
&Init \triangleq \\
&\quad \land\, lock = \text{TRUE} \\
&\quad \land\, coin \in \text{BOOLEAN} \\
&\quad \land\, push \in \text{BOOLEAN}
\end{aligned}
$$

$$
\begin{aligned}
&Step(c,\, p,\, x) \triangleq \\
&\quad \land\, lock' = ((\neg coin \land lock) \lor x) \\
&\quad \land\, coin' = c \\
&\quad \land\, push' = p
\end{aligned}
$$

$$
\begin{aligned}
&Next \triangleq \\
&\quad \exists\, c,\, p \in \text{BOOLEAN} : \\
&\quad \exists\, x \in \{(push \land \neg lock),\, \neg(push \land \neg lock)\} : \\
&\quad Step(c,\, p,\, x)
\end{aligned}
$$

$$vars \triangleq \langle lock,\, coin,\, push \rangle$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

$$
\begin{aligned}
&TypeInvariant \triangleq \\
&\quad \land\, lock \in \text{BOOLEAN} \\
&\quad \land\, coin \in \text{BOOLEAN} \\
&\quad \land\, push \in \text{BOOLEAN}
\end{aligned}
$$

THEOREM $Spec \Rightarrow \Box\, TypeInvariant$

$$S1 \triangleq \neg coin \land lock \Rightarrow lock'$$

$$Safety \triangleq \Box[S1]_{vars}$$

THEOREM $Spec \Rightarrow Safety$

Fig. 5. TLA+ specification for the out-of-nominal turnstile.

B Boolean Network Model for Turnstile in NuSMV

As described in Sect. 4.2, the NuSMV model checker is used to verify the robustness of the tiered combinational logic implementing the safety-critical term $\neg C \land L$, along the lines of previous work [12]. The inputs are taken as node 0 ($\neg C$) and node 1 (L), and the output is taken as node 18. The Boolean network (BN) is checked for conformance to the abstraction in the presence of any single internal bit flip in one of the first $n_{\max}$ tiers, where $n_{\max} \in \{1, \ldots, 20\}$. Figure 6 shows an extract from the model in the case where node 2 can be flipped

```
-- ...
  init(n02) := 0ub1_0;

  init(xfer02) := 0ub4_0011;

  init(nfn02) := 0ub1_0;
  init(latchn02) := FALSE;
  init(flipn02) := {FALSE, TRUE};

  next(flipn02) := {FALSE, TRUE};
  next(latchn02) := flipn02 | latchn02;
-- (flipn02 & (! latchn02)) happens at most once

-- xfern02 is the static transfer function for node 02
  next(n02) := ((0 <= tier) & (tier < 14) &
    (flipn02 & (! latchn02))) ? (! xfern02) : xfern02;

-- nfn02 is here to keep track of what the non-flipped bit would be
  next(nfn02) := xfern02;
-- ...
-- Property to be verified:
LTLSPEC F  ((tier = 20) & (n18 = (n00 & n01)));
```

Fig. 6. Extract from a NuSMV model that is programmatically generated so that all tiers and all nodes can be checked for susceptibility to bit flips. The linear temporal logic (LTL) property at the end expresses conformance of the out-of-nominal output to the abstraction $\neg C \wedge L$.

and $n_{\max} = 14$. It is found that the quiescent BN is immune to any single bit flip up to $n_{\max} = 15$, whereas the chaotic BN can be corrupted by a single bit flip for any value of $n_{\max}$.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, Cambridge (2010)
2. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: an opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
3. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**, 752–794 (2003)
4. Fey, G.: Assessing system vulnerability using formal verification techniques. In: Kotásek, Z., Bouda, J., Černá, I., Sekanina, L., Vojnar, T., Antoš, D. (eds.) MEMICS 2011. LNCS, vol. 7119, pp. 47–56. Springer, Heidelberg (2012)
5. Güdemann, M., Ortmeier, F.: Probabilistic model-based safety analysis. In: Proceedings of the 8th Workshop on Quantitative Aspects of Programming Languages, pp. 114–128, March 2010

6. Jackson, M., Zave, P.: Deriving specifications from requirements: an example. In: Proceedings of the 17th International Conference on Software Engineering, pp. 15–24 (1995)
7. Joshi, A., Heimdahl, M.P.E., Miller, S.P., Whalen, M.W.: Model-based safety analysis. NASA Contractor Report CR-2006-213953, February 2006
8. Joshi, A., Miller, S.P., Whalen, M., Heimdahl, M.P.: A proposal for model-based safety analysis. In: Proceedings of the 24th Digital Avionics Systems Conference, October 2005
9. Kauffman, S.A.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, Oxford (1993)
10. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 222–233, June 2011
11. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, Boston (2002)
12. Mayo, J.R., Armstrong, R.C., Hulette, G.C.: Digital system robustness via design constraints: the lesson of formal methods. In: Proceedings of the 9th Annual IEEE International Systems Conference, pp. 109–114, April 2015
13. Mytkowicz, T., Diwan, A., Bradley, E.: Computer systems are dynamical systems. Chaos **19**, 033124 (2009)
14. Seshia, S.A., Li, W., Mitra, S.: Verification-guided soft error resilience. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1442–1447, April 2007
15. Vorobeychik, Y., Mayo, J.R., Armstrong, R.C., Ruthruff, J.R.: Noncooperatively optimized tolerance: decentralized strategic optimization in complex systems. Phys. Rev. Lett. **107**, 108702 (2011)