SANDIA REPORT

SAND2014-1533
Unlimited Release
Printed January 2014

Dynamic Analysis Methods for Detecting Anomalies in Asynchronously Interacting Systems

Akshat Kumar, Benjamin Matschke, John Solis

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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 under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2014-1533
Unlimited Release
Printed January 2014

Dynamic Analysis Methods for Detecting Anomalies in Asynchronously Interacting Systems

Akshat Kumar
John Solis

Sandia National Laboratories
Livermore, CA, USA
akskuma@sandia.gov
jhsolis@sandia.gov

Benjamin Matschke

Max Planck Institute for Mathematics
Bonn, Germany
matschke@mpim-bonn.mpg.de

Abstract

Detecting modifications to digital system designs, whether malicious or benign, is problematic due to the complexity of the systems being analyzed. Moreover, static analysis techniques and tools can only be used during the initial design and implementation phases to verify safety and liveness properties. It is computationally intractable to guarantee that any previously verified properties still hold after a system, or even a single component, has been produced by a third-party manufacturer.

In this paper we explore new approaches for creating a robust system design by investigating highly-structured computational models that simplify verification and analysis. Our approach avoids the need to fully reconstruct the implemented system by incorporating a small verification component that dynamically detects for deviations from the design specification at run-time.

The first approach encodes information extracted from the original system design algebraically into a verification component. During run-time this component randomly queries the implementation for trace information and verifies that no design-level properties have been violated. If any deviation is detected then a pre-specified fail-safe or notification behavior is triggered.

Our second approach utilizes a partitioning methodology to view liveness and safety properties as a distributed decision task and the implementation as a proposed protocol that solves this task. Thus the problem of verifying safety and liveness properties is

translated to that of verifying that the implementation solves the associated decision task. We develop upon results from distributed systems and algebraic topology to construct a learning mechanism for verifying safety and liveness properties from samples of run-time executions.

Acknowledgments

This work was supported by both the Integrated Codes element of the Advanced Simulation and Computing program, and the Laboratory Directed Research and Development (LDRD) program at Sandia National Laboratories. National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Contents

1 Introduction and Motivation

Digital systems and distributed algorithms are quintessential compositions of asynchronously interacting components that communicate information to jointly perform desired computations over short runs. However, even if the components are themselves well-behaved — satisfying some local specifications — their individual computations or even the properties of their interacting communications (timing delays, etc.) may invalidate the joint computation by breaching safety properties. Worse yet, joint computations may not terminate if liveness properties are breached (i.e., by entering a non-halting execution path).

These issues may be avoided if the system design or specification is *validated* at early stages of development due to the availability of complete (theoretical) knowledge. However, if the implementation is carried out by an unverified third party, such as, a software compiler or a hardware manufacturer, then the lack of analogous knowledge of the system's implementation means that problems can still arise during run-time. In other words, there is still a concern about the final system's *verifiability* with respect to the original design.

Untrusted system implementations, particularly of manufactured integrated circuits, can be inspected using side-channel analysis [1, 5] and statistical logic testing techniques [3, 25] to detect for the presence of (malicious) modifications to system components. In general, these approaches are probabilistic by nature and cannot guarantee that all modifications are detected. Instead, a partitioning methodology that divides the original system into distinct components that jointly perform operations while keeping their individual inputs and computations private seems amenable to deterministic verifications. For example, a system based on cryptographically secure multi-party computation (MPC) primitives [4] could ensure that computations are performed correctly – even in the presence of malicious components. Unfortunately, MPC solutions are only suitable for specific classes of functions and cannot always be implemented efficiently, e.g., in hardware requiring register shift operations. However, we can borrow the core idea – multiple components performing operations over private states and information – to design a system that is robust to modifications in the implementation.

In this paper we explore new approaches for creating a robust system design that avoids complexity issues of earlier techniques by investigating highly-structured computational models that simplify verification and analysis. Our approach avoids the need to fully reconstruct the implemented system by incorporating a small verification component that dynamically detects for deviations from the design specification at run-time.

The first approach encodes information from the original system design into an algebraic structure derived from modeled trace information that can be embedded within a verification component. During run-time this component randomly queries the implementation for trace information and verifies that no design-level properties have been violated. If any deviation is detected then a pre-specified fail-safe or notification behavior can be triggered.

Our second approach utilizes the partitioning methodology to view the safety and liveness

properties of a system as a distributed decision task and the implementation as a proposed protocol that solves this task. Thus the problem of verifying safety and liveness properties is translated to that of verifying that the implementation solves the associated decision task. This approach is particularly promising because the latter problem has received much attention from both the distributed systems and mathematical communities [9, 19]. We develop upon these results to construct a learning mechanism for verifying safety and liveness properties from samples of run-time executions.

2 Preliminaries

We now review some preliminary concepts from distributed systems and provide formal descriptions of the models referenced in subsequent sections. We also describe how these models, for a given level of abstraction, accurately model the types of digital systems and distributed algorithms that we are interested verifying.

2.1 Asynchronous Transition Systems

Traditional models of computation, such as Turing Machines, are universal in the class of *sequential* systems, but lack the structure necessary to facilitate reasoning about the properties of complex distributed and asynchronous systems. Instead we must work with alternative computation models that capture the notion of asynchronicity between individual components of a larger system. Although several models have been proposed [21], we focus on the Asynchronous Transition System (ATS) [24] model; a generic but highly structured model to analyze interacting concurrent systems. In practice, these models are used to study the behavior of everything from computer networks to inter-connected hardware modules in integrated circuits.

We begin by formally introducing the concept of an asynchronous transition system and the notion of *morphisms* between two such systems. This will allow us to construct a category to abstract away much of the complexity so that we can reason about its high-level behavior. In particular, this structure allows the manifestation of notions of *bisimulations* among asynchronous systems [17, 21]. Deviations across specific implementations will manifest themselves in different ways at this higher level of abstraction. Initially, our only concern is detecting that a deviation exists. If a deviation exists then the entire system is discarded or replaced with a new implementation.

Definition 1. An *asynchronous transition system* (ATS) is a tuple $(S, s_0, E, I, Tran)$ where:

- S is the set of states;
- $s_0 \in S$ is an initial state;
- E is the set of transition labels (alphabet);
- $I \in E \times E$ is the *independency* relation among transitions, with the properties:
  - *irreflexivity*: given any $a \in E$, $(a,a) \notin I$ and
  - *symmetric*: if $(a,b) \in I$ then also $(b,a) \in I$

  and we denote the relationship $(a,b) \in I$ by $a \cong b$;
- $Tran \subseteq S \times E \times S$ is the relation of labelled transitions among states, wherein we denote $(s,e,s') \in Tran$ by $s \xrightarrow{e} s'$.

Moreover, the following conditions must be satisfied by these components:

Movement condition: The "0" transition is unique and given any $e \in E$, there exist $s, s' \in S$ such that $s \xrightarrow{e} s'$.

Well-definedness: The relation $Tran$ is a *partial map* $Tran : S \times E \rightharpoonup E$, in that

$$s \xrightarrow{e} s', s \xrightarrow{e} s'' \Rightarrow s' = s''.$$

Independencies are commutative transitions: Given any $(a, b) \in I$, if $s, s_1, s_2 \in S$ are taken such that $s \xrightarrow{a} s_1 \xrightarrow{b} s_2$, then there exists $s_1' \in S$ such that $s \xrightarrow{b} s_1' \xrightarrow{a} s_2$.

Remark 1. If $\circ$ denotes function composition, then the last condition can also be written as $a \cong b \Rightarrow a \circ b = b \circ a$, though care should be taken to ensure that the domain and co-domains of a and b allow for valid function composition.

2.2 Morphisms of Asynchronous Transition Systems

Morphisms describe how to construct a mapping between two different transition systems. Intuitively, we want to construct a mapping between each of the objects of an ATS tuple and provide appropriate conditions to couple the different maps in a natural way:

Definition 2. A morphism of asynchronous transition systems is a mapping $(\sigma : S \to S', \eta : E \rightharpoonup E') : (S, s_0, E, I, Tran) \to (S', s_0', E', I', Tran')$ such that σ is *based*, viz., $(s_0 \mapsto s_0')$ and the following conditions are sufficed:

$$\begin{cases} s \xrightarrow{a} t \Rightarrow \sigma(s) \xrightarrow{\eta(a)} \sigma(t) & \text{if } a \in Dom(\eta) \\ \sigma(s) = \sigma(t) & \text{otherwise} \end{cases}$$
$$\forall e_1, e_2 \in Dom(\eta), e_1 \underset{I}{\cong} e_2 \Rightarrow \eta(e_1) \underset{I'}{\cong} \eta(e_2)$$

Note that in our definition of a morphism it is not necessary for η to be defined everywhere on E — partially defined maps are also acceptable. The notation $\underset{I}{\cong}$ refers to independence relations defined according to I.

2.3 $\mathcal{A}$ Category

We can now use the previous definitions to construct the category $\mathcal{A}$ of ATS's in a straightforward manner: $\mathrm{Obj}(\mathcal{A})$ is the collection of asynchronous transition systems $(S, s_0, E, I, Tran)$ and for any two systems S and S',

$$\mathrm{Hom}(S, S') \coloneqq \{(\sigma : S \to S', \eta : E \rightharpoonup E')\}$$

is the collection of ATS morphisms among them.

3 From *ATS* to Monoidal Structures

Our current definition of an ATS includes a lot of information about the structure of the underlying system. However, it can still be difficult to reason about properties, e.g., liveness and safety, when using this model. To help analyze for properties of interest, we translate our problem from an asynchronous transition system into a more algebraic, monoidal structure.

3.1 Structure for a Monoid Action

An *augmented* ATS structure adds a *terminal* element $*$ to the state space of a given ATS, allowing us to translate ATS structures into monoid actions, by ensuring that all transitions are maps (and not just *partial* maps). Let $S_* \triangleq S \sqcup \{*\}$ and for a given event $e \in E$, define (by a slight abuse of notation) $e : S_* \to S_*$ by

$$* \mapsto *$$
$$s \mapsto \begin{cases} s' & \text{if } s \xrightarrow{e} s' \ (s,e,s') \in Tran \\ * & \text{otherwise} \end{cases}.$$

We can thus collect these maps to get a map over E, in the natural way:

$$\begin{aligned} \widetilde{S} : S_* \times E &\to S_* \\ (s,e) &\mapsto e(s). \end{aligned}$$

Note that for every $e_1, e_2 \in E$, $e_1 \circ e_2 : S_* \to S_*$. This can be extended by induction to any finitely many $e_1, ..., e_n \in E$. Also note that $e_1 \underset{I}{\cong} e_1 \Rightarrow e_1 \circ e_2 = e_2 \circ e_1$, whence all of the independence relations are preserved in the augmented object.

3.2 Monoid Actions and the associated Category

Given an ATS with transition alphabet E and independency relations I, let E^* be the free monoid on E and I^* the transitive completion of I. Then, E^*/I^* is a partially commutative free monoid on E where the identity element is the empty word ε. This motivates the following introduction of the notion of a *monoid action* of a given monoid on a given set:

Definition 3. A *(right-) monoid action* of a monoid M on a set X is a map $\cdot : X \times M \to X$ satisfying:

$$s \cdot t = \begin{cases} s & \text{if } t = \varepsilon \\ (s \cdot a) \cdot b & \text{if } t = ab \end{cases}$$

We will denote a set X equipped with a monoid action of M by a tuple, (M, X). In particular, for the monoid $\mathcal{M} = E^*/I^*$ we will define the action $\cdot : S_* \times \mathcal{M} \to S_*$ on S_* by

$$s \cdot (e_1 ... e_n) = (e_1 \circ \cdots \circ e_n)(s).$$

Definition 4. Let X_*, X'_* be two pointed sets equipped with monoid structures (M, X_*) and (M', X'_*), respectively. Then,

$$(\eta : M \to M', \sigma : X_* \to X'_*) : (M, X_*) \to (M', X'_*)$$

is a *morphism* among monoid actions if for all $a, b \in M$,

$$\eta(a \underset{M}{\cdot} b) = \eta(a) \underset{M'}{\cdot} \eta(b)$$

and for all $x \in X, \mu \in M$,

$$\sigma : \begin{cases} * \mapsto * \\ x \underset{M}{\cdot} \mu \mapsto \sigma(x) \underset{M'}{\cdot} \eta(\mu) \end{cases}$$

Basically, a morphism between monoid actions preserves the actions: monoid actions of M on X_* and M' on X'_*, respectively. In particular, it is easy to see that any ATS-morphism $(\eta : E \rightharpoonup E', \sigma : S \to S') : (S, s_0, E, I, Tran) \to (S', s'_0, E', I', Tran')$ naturally induces a monoid morphism $(\eta^* : E^* \to E'^*, \sigma : S_* \to S'_*) : (\mathcal{M}, S_*) \to (\mathcal{M}', S'_*)$.

We can also now define a categorical structure, $Cat(M, X)$, associated with the structure (M, X), by taking $\mathrm{Obj}(M, X) = X$ and $\mathrm{Hom}(s \to t) \coloneqq \{\omega \in M \mid s \cdot \omega = t\}$. In fact, this structure is also a two-category, meaning that the objects themselves are categories and morphisms are functors between categories, since every monoid itself is a pointed category. Indeed, $Cat(M, X)$ is a natural generalization of this idea.

4 Algebraic Data on an ATS

4.1 Simplicial Sets

Let *Ord* denote the category of finite ordered sets with monotonic maps as morphisms, i.e., maps $\overline{n} \to \overline{m}$, where $\overline{n} := \{0, ..., n\}$. To each $S \in \text{Obj}(Ord)$ there naturally corresponds a simplex $\Delta(S)$. This leads us to consider two types of functors on *Ord*:

1. The *covariant* functors $\Delta : Ord \to Top$ mapping into the category of topological spaces with continuous functions as morphisms, which preserve the orders in morphisms:
$$\overline{n} \overset{\Delta}{\to} A_n, \overline{m} \overset{\Delta}{\to} A_m \Rightarrow (\overline{n} \to \overline{m}) \overset{\Delta}{\mapsto} (A_n \to A_m)$$
and map each $S \in \text{Obj}(Ord)$ to a simplex with vertices in S.
2. The *contravariant* functors $\mathcal{F} : Ord \to Set$ mapping in an order-reversing fashion into the category of sets, i.e.,
$$\overline{n} \overset{\mathcal{F}}{\to} A_n, \overline{m} \overset{\mathcal{F}}{\to} A_m \Rightarrow (\overline{n} \to \overline{m}) \overset{\mathcal{F}}{\mapsto} (A_m \to A_n).$$

Thus, $\mathcal{F}$ induces boundary maps $A_n \to A_{n-1} \to \cdots \to A_1 \to A_0$ and we can realize a triangulated geometric structure associated with S in the following way:

Definition 5. Let $\mathcal{O} := \text{Obj}(Ord)$. Then, we define $|\Delta(\mathcal{F})| := \Big(\bigsqcup_{S \in \mathcal{O}} \Delta(S) \times \mathcal{F}(S) \Big) \Big/ \sim$, where for $\theta \in \text{Hom}_{Ord}(S, T)$, $x \in S$, and $a \in T$, we make the relation $(x, \mathcal{F}(\theta)(a)) \sim (\Delta(\theta)(x), a)$.

Now, note that each ordered set S is in itself a category as well, in the sense that we can regard each element of S to be an object and the ordering of elements to be the unique morphisms. So, given any other category $\mathbb{C}$, each functor $S \to \mathbb{C}$ gives an ordering among objects in $\mathbb{C}$ through morphisms among those objects, giving rise to *chains* of morphisms in $\mathbb{C}$. Then, taking any other $T \in \text{Obj}(Ord)$, since each functor $\theta : S \to T$ is just a monotonic map in $\text{Hom}(S, T)$, θ gives rise to maps among morphism chains in $\mathbb{C}$ corresponding to S and T. This motivates the following:

Definition 6. Let $\mathbb{C}$ be a *small* category, meaning that $\text{Obj}(\mathbb{C})$ and $\text{Hom}(\mathbb{C})$ are both sets, and let $hom(\mathbb{C}', \mathbb{C})$ denote the space of functors $\mathbb{C}' \to \mathbb{C}$. The *nerve* of $\mathbb{C}$ is the functor $\mathcal{N}(\mathbb{C}) : Ord \to Set$ given by:
$$\mathcal{N}(\mathbb{C})(S) = hom(S, \mathbb{C})$$
$$\mathcal{N}(\mathbb{C})(S \to T) = \{hom(T, \mathbb{C}) \to hom(S, \mathbb{C})\}.$$

Combining the structures in definitions 6 and 5, we have that
$$|\Delta(\mathcal{N}(\mathbb{C}))| = \Big(\bigsqcup_{S \in \text{Obj}(Ord)} \Delta(S) \times \mathcal{N}(\mathbb{C})(S) \Big) \Big/ \sim .$$
This process turns our chains into a simplicial complex structure.

4.2 Homology

The image of $\overline{n}$ under $\mathcal{N}(\mathbb{C})$ encodes a sequence of n composable maps $c_0 \overset{\alpha_1}{\to} c_1 \overset{\alpha_2}{\to} \cdots \overset{\alpha_n}{\to} c_n$ for $n > 0$ and the set of all objects $c_0 \in \mathrm{Obj}(\mathbb{C})$ for $n = 0$. By this and the basic properties of morphisms in categories, we can define the following operators:

Definition 7. The *face operators* $d_i^n : \mathcal{N}\mathbb{C}(\overline{n}) \to \mathcal{N}\mathbb{C}(\overline{n-1})$, with $0 \leq i \leq n$, are defined as follows:

when $0 < i < n$:

$$\begin{array}{c} c_0 \overset{\alpha_0}{\to} c_1 \overset{\alpha_1}{\to} \cdots \overset{\alpha_{i-2}}{\to} c_{i-1} \overset{\alpha_{i-1}}{\to} c_i \overset{\alpha_i}{\to} c_{i+1} \overset{\alpha_{i+1}}{\to} \cdots \overset{\alpha_{n-1}}{\to} c_n \\ \downarrow \\ c_0 \overset{\alpha_0}{\to} c_1 \overset{\alpha_1}{\to} \cdots \overset{\alpha_{i-2}}{\to} c_{i-1} \overset{\alpha_{i-1}\circ\alpha_i}{\to} c_{i+1} \overset{\alpha_{i+1}}{\to} \cdots \overset{\alpha_{n-1}}{\to} c_n \end{array}$$

when $i = 0$:

$$\begin{aligned} d_0^n(c_0 \to c_1 \to \cdots \to c_n) \\ = c_1 \to \cdots \to c_n \end{aligned}$$

when $i = n$:

$$\begin{aligned} d_n^n(c_0 \to c_1 \to \cdots \to c_n) \\ = c_0 \to \cdots \to c_{n-1}. \end{aligned}$$

That is, an (i,n)-face operator removes the ith object from every n-chain. Dually, we have:

Definition 8. The *degeneracy operators* $\mathbb{S}_i^n : \mathcal{N}(\mathbb{C})(\overline{n}) \to \mathcal{N}(\mathbb{C})(\overline{n+1})$, with $0 \leq i \leq n$, are given by

$$\begin{array}{c} c_0 \overset{\alpha_0}{\to} c_1 \overset{\alpha_1}{\to} \ldots \overset{\alpha_{i-2}}{\to} c_{i-1} \overset{\alpha_{i-1}}{\to} c_i \overset{\alpha_i}{\to} c_{i+1} \overset{\alpha_{i+1}}{\to} \ldots \overset{\alpha_{n-1}}{\to} c_n \\ \downarrow \\ c_0 \overset{\alpha_0}{\to} c_1 \overset{\alpha_1}{\to} \cdots \overset{\alpha_{i-2}}{\to} c_{i-1} \overset{\alpha_{i-1}}{\to} c_i \overset{id}{\to} c_i \overset{\alpha_i}{\to} c_{i+1} \overset{\alpha_{i+1}}{\to} \cdots \overset{\alpha_{n-1}}{\to} c_n, \end{array}$$

where id is the identity map.

These operators provide the categorical interpretation of the associated natural face and degenracy maps on the complex $\Delta(\mathcal{N}(\mathbb{C}))$. Indeed, it is an easy exercise to verify that the usual *simplicial identities* are satisfied by these maps.

Now, given any abelian group G and any $n \geq 0$, we can take formal G-linear combinations of chains in $\mathcal{N}(\mathbb{C})(\overline{n})$, forming the *chain group* $C_n(|\Delta(\mathcal{N}(\mathbb{C}))|; G)$, which we will denote simply by C_n for succinctness (context permitting). The face maps above naturally give rise to *boundary* homomorphisms $\partial_n : C_n \to C_{n-1}$, such that $\partial_{n+1} \circ \partial_n = 0$, and as usual, we define the *homology groups* $H_n := \ker \partial_n / \operatorname{im} \partial_{n+1}$.

If we take $\mathbb{C} = Cat(\mathscr{M}, S_*)$, with respect to some ATS, then the homology groups encode information about the concurrency/asynchronicity properties of the system. We call the largest $n \geq 0$ such that $H_n \neq \{0\}$ the *homological dimension* of the ATS. It was shown in [13] that the homological dimension n of an ATS is bounded above by the number of pairwise independencies in the system. Moreover, through the construction of an *adjoint* functor between the categories of *higher-dimensional automata* and asynchronous transition systems [7, 15], we can associate an asynchronous process calculus with algebraic operations on the homology groups, as in [8, 18]. The lack of isomorphicity among homology groups also forms an *obstruction* to bisimulation equivalence, in the sense of [17], among ATSs [16].

5 Dynamic Homology Analysis

5.1 The General Setup

Let $M(\Sigma, I)$ be the quotient of the free monoid Σ^* on a non-empty finite set Σ, by the closure of the irreflexive and symmetric binary relation $I \subseteq \Sigma \times \Sigma$; in other words, $M(\Sigma, I)$ is given by the monoid presentation,

$$M(\Sigma, I) = \langle \Sigma \,|\, ab = ba, \forall (a,b) \in I \rangle.$$

The elements of $M(\Sigma, I)$ are called *traces* and $M(\Sigma, I)$ itself is called a *trace monoid* with *alphabet* Σ (consisting of *letters*) and *independence relation* I. Given a non-unit trace ω, the unique sequence $(a_1, \ldots, a_k)$ of letters such that $a_1 \cdots a_k = \omega$ is called the *decomposition* of ω, and we denote by ε the unit element. Now, for a finite set S, which we will call the *set of states*, we have the following strengthening of the notion of a *strong partial* monoid action (i.e., in the sense of [12]) of $M(\Sigma, I)$ on S by a well-definedness condition on the traces:

Definition 9. Let $Par(S)$ be the set of all partial functions on S with non-empty domains of definition and $\iota : S \to S$ the identity map. Then, a *partial monoid action* of $M(\Sigma, I)$ on S, *well-defined on* $M(\Sigma, I)$, is a function $\cdot : M(\Sigma, I) \to Par(S)$, for which we denote the application of $\omega \mapsto \omega\cdot$ by $\omega \cdot s := (\omega\cdot)(s)$ whenever $s \in \mathrm{dom}(\omega\cdot)$, and that satisfies the conditions:

$$\varepsilon\cdot = \iota; \tag{PA1}$$

$$s \in \mathrm{dom}(\omega\cdot) \Longrightarrow [s \in \mathrm{dom}(\mu\omega\cdot) \iff \omega \cdot s \in \mathrm{dom}(\mu\cdot), \text{ in which case, } \mu\omega \cdot s = \mu \cdot (\omega \cdot s)]. \tag{PA2}$$

We use the notation $M(\Sigma, I) \cdot S$ to denote a partial monoid action well-defined on $M(\Sigma, I)$.

On adjoining a singleton $\{*\}$ to S, we see that the partial monoid action extends to a monoid action by setting $\omega \cdot s = *$ whenever $s \notin dom(\omega\cdot)$, for each $\omega \in M(\Sigma, I)$ and $s \in S \sqcup \{*\}$. We shall set $S_* := S \sqcup \{*\}$ for brevity and continue to denote the monoid action in the same way as the partial monoid action. Associated with this monoid action (and so to the partial monoid action) are the categories $\mathcal{K}(M(\Sigma, I) \cdot S_*)$ and $\mathcal{K}(M(\Sigma, I) \cdot S)$, which in unambiguous situations we may simply denote by $\mathcal{K}_*$ and $\mathcal{K}$, respectively, where $\mathcal{K}_*$ is defined by

$$Ob(\mathcal{K}_*) = S_*;$$
$$Hom_{\mathcal{K}_*}(s, s') = \{\omega \in M(\Sigma, I) \,|\, \omega \cdot s = s'\}, (\forall) s, s' \in S_*$$

and $\mathcal{K}$ is given by removing the terminal $*$ from $Ob(\mathcal{K}_*)$ and all morphisms targeting it, from $Hom_{\mathcal{K}_*}$. Henceforth, we shall consider only the *unaugmented* category $\mathcal{K}$. Thus, we also naturally have a topological structure associated with $M(\Sigma, I) \cdot S$ and its monoid extension, given by the classifying space of $\mathcal{K}$ [22]. In the following, we will require that the classifying space is always connected, which is equivalent to the condition that the finite graph which represents the partial maps $\Sigma\cdot = \{\alpha\cdot \in Par(S) \,|\, \alpha \in \Sigma\}$ be connected.

Since the classifying space of $\mathcal{K}(M(\Sigma, I) \cdot S)$ has a natural simplicial set decomposition, we can use the usual face maps to generate a simplicial chain complex and thus compute its

homology groups. However, in general, the chain complex need not be finite nor have all chain groups be finitely generated. Thus, we instead rely on the semicubical construction of the chain complex, which is detailed in [13, 14] along with a proof in [14] that it will always yield a finite chain complex with finitely generated chain groups while preserving the homology. Finally, we will assume throughout that the base ring for the chain complexes is $\mathbb{Z}$ and we will denote the nth homology group for $\mathcal{K}(M(\Sigma, I) \cdot S)$ by $H_n(M(\Sigma, I) \cdot S)$ (or simply H_n, when the context is clear).

5.2 Testing Samples on the Space Against Test Homology Groups

Now, suppose we are given the set of states T and alphabet Ω of a *sample space* $\mathcal{K}(M(\Omega, I_\Omega) \cdot T)$, where the independence relation I_Ω and action $M(\Omega, I_\Omega) \cdot T$ are unknown. On this space, we are able to take samples of the action on T, which we call *trace samples*, that give us the transitions between points in S and the sequence of letters from Ω that compose the action. Given also a *test space* $\mathcal{K}(M(\Sigma, I) \cdot S)$, about which we have full information, we can decide whether or not it has the same asynchronicity properties as the sample space by monitoring the birth and death effects on the homologies of trace samples from the sample space and comparing these relative homologies with those of the test space. However, we would like to optimize and algorithmize this process through a procedure of answering the following questions:

1. *Does there exist a means of translating sequences of trace samples from the sample space into combinations of actions in the nth homology group of the test space?*
2. *If yes, then do there exist some computations involving these combinations of actions over the group, which can determine whether or not the nth homology groups of the sample space and test space are isomorphic?*
3. *If yes, then is it sufficient to take just one sequence of trace samples to determine the (lack of) isomorphicity (at each sample step)?*

The first two questions seem to be answerable in the positive by extending the methods of [2] and related work. The final question, however, requires the construction of an ideal sampling methodology.

6 Detecting Liveness and Safety Failures from Sampled Behaviours

Given a system composed of $N > 1$ communicating components, we can choose a class of asynchronous automata, such as Lynch-Tuttle's I/O automata [20], to provide a formal model for the system. Moreover, we can model the knowledge from the design about the components' initial states and the system's possible safe final states for some interval of time during which work is performed, as a distributed *decision task*. Such models are commonly adopted for reasoning about finite computations and protocols in multiprocessor systems and distributed algorithms, alike. In our case, we can take some *critical* states or sets of inputs for the system, where safety is of utmost concern. We assume validity of the design, from which it follows that the associated decision task is known (and required) to be solvable in some class of liveness concerns. Further, we assume that the only *a priori* data available about the implementation are the N components and a means of initializing the system to the given initial component states, but that during run-time it is possible to attain the full state of the system at any point in time. Since such state *snapshots* are available, we can assume, without any loss of generality, that communications among the components are governed by the *atomic snapshot model* (see [19] for the generality of this memory model), purely for the purposes of analysis.

Since our considerations include timing concerns of events occuring from the interacting components, such as communications and real-time executions over shared resources, we face the problem of lacking a universal model for such concurrent computations, analogous to that of the Turing machine (or its suitable restrictions) for sequential systems. This kind of divergence in universality is further exacerbated when liveness properties are taken into account: Fischer, Lynch and Patterson [6] have shown a Turing-computable task that is not asynchronously solvable in the presence of even one failing component. Accordingly, we must rely on a modeling hierarchy that exhibits the classes of liveness properties that we are concerned about. In the sequel, we will focus on the consistent global progression of asynchronous compositions with varying liveness concerns, starting with the strongest assumptions — the *wait-free* case: possible failure of all but one component — and by means of simulated reductions [10], extend our results to weaker assumptions as well.

Thus, the problem of verifying a given implementation of an asynchronous composition of interacting systems over given work segments with safety and liveness concerns coming from a valid specification, can be recast as that of checking whether the automata model representing the implementation solves the associated decision task. This reformulation is further elaborated upon in § 6.1. Focusing firstly on the case where liveness falls in wait-freedom, we rely on the seminal work of Herlihy and Shavit [11], which characterizes solubility of wait-free decision tasks by a combinatorial-topological condition. Working with this operational model and topological setup, we will construct a topological obstruction theory to characterize the detectability of failures in solving a given wait-free decision task from samples of executions. This, along with combinatorial arguments, informs us about viable classes of samples needed to develop a mechanism for attaining useful samples in

practice.

6.1 Obstructions to Wait-Free Solvability

Reformulation into a Distributed Decision Task Problem

Throughout, we will follow the setup of [11], in which a distributed system of communicating parties is modeled by a simplified I/O automaton, which effectively preserves just the communications data among components over a read-write memory model (particularly, atomic snapshot memory). Following the language of that paper, we call such an automaton a *protocol*. Further, a tuple $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is called a *decision task*, if the *input complex* $\mathcal{I}$ and the *output complex* $\mathcal{O}$ are the simplicial complexes representing the collection of input and output vectors, respectively, and $\Delta \subseteq \mathcal{I} \times \mathcal{O}$ is a multivalued mapping $\mathcal{I} \to \mathcal{O}$ that preserves the number of participating processes (equivalently, it preserves simplicial dimensions), called the *task specification*. Then, a protocol is said to *solve* a decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ if every process halts after finitely many steps and the collection of final outputs (if any) respects the relation Δ.

Let $\mathscr{P}$ be a system composed of $N > 1$ communicating components accomplishing some finite computations together, and let its safety requirements, determined from the design (i.e., via model-checking or simulation), be given in the form of a (finite) collection of initial states $\mathcal{I}$ for the components and corresponding (finitely many) possible safe final states $\mathcal{O}$ for the given work segments. It is, of course, implicitly assumed that each state vector in $\mathcal{I}$ and in $\mathcal{O}$ belongs to some global state for $\mathscr{P}$. The safety specifications clearly form a task specification Δ the sense of [11], so that $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is a decision task. We wish to formally equate the problem of verifying $\mathscr{P}$ according to specifications with checking if an associated protocol solves this distributed decision task over shared read/write memory. To this end, we provide each component P of $\mathscr{P}$ with a *harness*, which writes each state evolution of P to shared memory and reads from memory the states of all other harness components, passing along to P any outputs designated for it by other components and absorbing any outgoing communications from P otherwise (this is possible since message passing is determined by a local state evolution in each component). Since communications among components are unaltered and any delays or failures in components are mimicked by the harnesses, the functionality and safety properties of the original asynchronous network are left unaltered. Therefore, it is straightforward to view this harness model as a protocol in the above sense, whence we have the intended reformulation of the original verification problem. In practical terms, the harnessed setup simply models the tracing of observable states of $\mathscr{P}$, i.e. through a debugging apparatus. Of course, whenever the final states are determined completely by component outputs, we need not go to the additional setup as it suffices to simply view all communications as read/write operations over an atomic snapshot memory object, as per [11].

Setup for Obstruction Theory

Given a decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ and a simplex S in a subdivision of $\mathcal{I}$, the *carrier* of S in $\mathcal{I}$, denoted by $K(S, \mathcal{I})$, is the minimal simplex in $\mathcal{I}$ containing S. With this, the celebrated Asynchronous Computability Theorem (ACT) of Herlihy and Shavit [11] states:

Theorem 1 (ACT)**.** *A decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a wait-free protocol using read-write memory if and only if there exists a chromatic subdivision $\sigma(\mathcal{I})$ of $\mathcal{I}$ and a color-preserving simplicial map*

$$\mu : \sigma(\mathcal{I}) \to \mathcal{O}$$

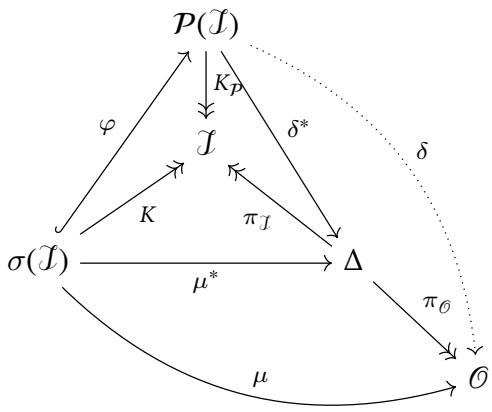
such that for each simplex S in $\sigma(\mathcal{I})$, $\mu(S) \in \Delta(K(S, \mathcal{I}))$.

The *protocol complex* $\mathcal{P}(\mathcal{C})$ corresponding to a protocol $\mathcal{P}$ and input subcomplex $\mathcal{C}$ for a given I/O pair $\langle \mathcal{I}, \mathcal{O} \rangle$ is the subcomplex formed from the collection of possible outputs for executions from $\mathcal{P}$ initiated with inputs from $\mathcal{C}$. It is easily seen that a protocol $\mathcal{P}$ wait-free solves a decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ if and only if there exists a color-preserving simplicial map

$$\delta : \mathcal{P}(\mathcal{I}) \to \mathcal{O},$$

called the *decision map*, with the property that $\delta(\mathcal{P}(S)) \subseteq \Delta(S)$ for every simplex S in $\mathcal{I}$. Moreover, if there is a color-preserving simplicial map $\varphi : \sigma(\mathcal{I}) \to \mathcal{P}(\mathcal{I})$ that maps every simplex $S \in \sigma(\mathcal{I})$ into $\mathcal{P}(K(S, \mathcal{I}))$, called the *span*, then $\mu = \delta \circ \varphi$ clearly satisfies ACT above.

Let Δ^* be the pure full-dimensional *polyhedral* subcomplex of $\mathcal{I} \times \mathcal{O}$ that has a facet (S, T) for every facet S of $\mathcal{I}$ for which T is in $\Delta(S)$. Indeed, the purity and full-dimensionality of Δ^* are immediate from the fact that these properties hold for $\mathcal{I}$ and $\mathcal{O}$ and that Δ preserves simplicial dimensions. Since Δ^* is only a geometric reformulation of Δ and holds the same information, we will from now on use them interchangeably. Now, given $\delta : \mathcal{P}(\mathcal{I}) \to \mathcal{O}$, we define $\delta^* : \mathcal{P}(\mathcal{I}) \to \Delta$ by the property that for every simplex S in $\mathcal{I}$, $\delta^*(\mathcal{P}(S)) \subseteq (S, \Delta(S))$. Similarly we can lift μ to $\mu^* : \sigma(\mathcal{I}) \to \Delta$. Now, let $\pi_{\mathcal{I}} : \Delta \to \mathcal{I}$ and $\pi_{\mathcal{O}} : \Delta \to \mathcal{O}$ be the canonical projections onto $\mathcal{I}$ and $\mathcal{O}$, repsectively, and $K_{\mathcal{P}} : \mathcal{P}(\mathcal{I}) \to \mathcal{I}$ be the color-preserving simplicial carrier map with respect to $\mathcal{P}$, which sends $\mathcal{P}(S)$ to S for every simplex S of $\mathcal{I}$. Then, the content of [11] and the above remarks show that the wait-free solvability of a decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ by a protocol $\mathcal{P}$ is equivalent to the existence of a color-preserving simplicial $\delta : \mathcal{P}(\mathcal{I}) \to \mathcal{O}$ that induces the commutative diagram (3) below, which in turn is equivalent to the existence and commutativity of the portion of (3) with solid arrows (where μ, φ, and the carrier maps are taken to be simplicial and color-preserving).



(3)

In particular, we see that $\mathcal{P}$ wait-free solves the decision task if and only if there exists a subdivision $\sigma(\mathcal{I})$ and map $\mu^* : \sigma(\mathcal{I}) \to \Delta$, such that $\pi_{\mathcal{I}} \circ \mu^* = K$ is the carrier map and $\mu := \pi_{\mathcal{O}} \circ \mu^*$ is simplicial and color-preserving. Now, if $\pi_{\mathcal{I}}$ is a *fibration*, then such a μ^* is a *(cross-)section* of $\pi_{\mathcal{I}}$. Moreover, if μ_n is a partially defined section of $\pi_{\mathcal{I}}$ on the $k-$*skeleton* of $\sigma(\mathcal{I})$, then there exists an extension of μ_n to a $\sigma(\mathcal{I})$ if and only if the associated *obstruction cocycle* is trivial in the cochain group $C^{n+1}(\mathcal{I}, \pi_n(F))$, where $\pi_n(F)$ is the n-homotopy group on the fiber F of $\pi_{\mathcal{I}}$. Thus, we are able to ascertain the existence of errors in a given protocol whenever an n-cycle α appears in a sample of executions, such that α is not *null-homotopic*.

The fibration condition on $\pi_{\mathcal{I}}$ is really a topological condition on the task specification Δ and by an appropriate choise of cochain complexes, we can construct an obstruction theory that works for arbitrary specifications Δ, up to the basic requirements in [11]. As such, if A is a subspace of $\mathcal{I}$ given by samples of executions in the protocol complex, then under our assumptions, we have the following possibilities:

1. Any section over A is extendible to $\mathcal{I}$, meaning that there are no issues in any possible continuation of the protocol;
2. No section over A extends to $\mathcal{I}$, meaning that there is some flaw in every continuation of the protocol;
3. Some sections over A extend to $\mathcal{I}$ and some do not, which is the most interesting case in terms of samples.

In the case of possibility (2), we can further analyze subprotocols over A by a restriction of the appropriate maps, and for (3) we would like to characterize the minimal such A. Furthermore, we would like to *relativize* the theory such that when given a subspace B and a section $s|_B$ over it (i.e., the part of the protocol that we have already sampled), we would like to characterize the minimal subspaces A containing B such that some extensions of $s|_B$ to A have an extension to I. These expansions would allow us to better understand and optimize the modes of sampling to predictively ascertain faulty paths in the system.

Finally, we note that obstruction classes and cohomology groups are particularly amenable to algorithmic computations and by recent developments in Homotopy Type Theory [23] (HoTT), we see clear paths to formally verifiable fault certificates (i.e., in semi-automated

proof systems such as Coq or Agda) for interacting systems through the embedding of our obstruction theory in HoTT via Postnikov truncations.

7 Conclusion

In this paper we investigated new dynamic analysis methods for detecting modifications to asynchronous transition systems based on highly-structured computational models. Our approaches avoid exhaustive testing and the need to fully reconstruct the implemented system by incorporating a small trusted component into the post-manufactured system that dynamically queries the implementation.

The first approach encodes the generators of the homology groups extracted from the original system design into the trusted component. During run-time, the trusted component queries the specific implementation to extract a single trace sample and verifies that the returned trace can be computed by the reference generators. If any deviation from the original reference homology is detected, then a pre-specified fail-safe or notification behavior is triggered.

The second approach determines the existence of liveness and safety failures occurring in joint finite computations among a number of interacting components by analyzing samples of run-time executions using models of the system's asynchronous behaviours derived from its design-level information.

For both approaches we have outlined a basic sampling and testing algorithm and discuss how this information can be used to verify that the implementation meets the original specification. However, many open questions, particularly with regards to improving efficiency, still remain before this theory can be applied to practical systems.

References

[1] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan detection using ic fingerprinting. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 296–310. IEEE, 2007.

[2] Oleksiy Busaryev, Sergio Cabello, Chao Chen, Tamal K Dey, and Yusu Wang. Annotating simplices with a homology basis and its applications. In *Algorithm Theory–SWAT 2012*, pages 189–200. Springer, 2012.

[3] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. Mero: A statistical approach for hardware trojan detection. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 396–410. Springer, 2009.

[4] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation, an introduction. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[5] Hidde De Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*, 9(1):67–103, 2002.

[6] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[7] ERIC GOUBAULT. Geometry and concurrency: a user's guide. *Mathematical Structures in Computer Science*, 10:411–425, 8 2000.

[8] Eric Goubault and Thomas P Jensen. Homology of higher dimensional automata. In *CONCUR'92*, pages 254–268. Springer, 1992.

[9] M. Herlihy, P. Jayanti, S. Kutten, and DISC. *Distributed Computing: ... International Symposium ; Proceedings. Toledo, Spain, October 4-6, 2000.* Number v. 14 in Lecture notes in computer science. Springer, 2000.

[10] Maurice Herlihy and Sergio Rajsbaum. Simulations and reductions for colorless tasks. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 253–260. ACM, 2012.

[11] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM (JACM)*, 46(6):858–923, 1999.

[12] Christopher Hollings. Partial actions of monoids. *Semigroup Forum*, 75(2):293–316, 2007.

[13] A. A. Husainov. The Cubical Homology of Trace Monoids. *ArXiv e-prints*, October 2011.

[14] A. A. Husainov. The Homology Groups of a Partial Trace Monoid Action. *ArXiv e-prints*, November 2011.

[15] Ahmet A Husainov. Cubical sets and trace monoid actions. *The Scientific World Journal*, 2013, 2013.

[16] Ahmet A Husainov. Homology and bisimulation of asynchronous transition systems and petri nets. *arXiv preprint arXiv:1307.5377*, 2013.

[17] André Joyal, M Nielson, and Glynn Winskel. Bisimulation and open maps. In *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*, pages 418–427. IEEE, 1993.

[18] AA Khusainov, VE Lopatkin, and IA Treshchev. Studying a mathematical model of parallel computation by algebraic topology methods. *Journal of Applied and Industrial Mathematics*, 3(3):353–363, 2009.

[19] N.A. Lynch. *Distributed Algorithms*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 1996.

[20] Nancy Lynch and Mark Tuttle. An Introduction to Input/Output automata. Technical Memo MIT/LCS/TM-373, Massachusetts Institute of Technology, November 1988.

[21] Mogens Nielsen and Glynn Winskel. Petri nets and bisimulation. *Theoretical Computer Science*, 153(1):211–244, 1996.

[22] Graeme Segal. Classifying spaces and spectral sequences. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 34(1):105–112, 1968.

[23] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[24] Glynn Winskel and Mogens Nielsen. Models for concurrency. *DAIMI Report Series*, 22(463), 1993.

[25] Francis Wolff, Chris Papachristou, Swarup Bhunia, and Rajat Subhra Chakraborty. Towards trojan-free trusted ics: Problem analysis and detection scheme. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 1362–1365. IEEE, 2008.

DISTRIBUTION:

1	MS 9158	John H. Solis, 8961
1	MS 9158	Akshat Kumar, 8961
1	MS 9158	Keith Vanderveen, 8961
1	MS 9152	Robert Clay, 8953
1	MS 0899	Technical Library, 9536 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office, 1911

Sandia National Laboratories