Contract-based Blame Assignment by Trace Analysis

Shaohui Wang
University of Pennsylvania
shaohui@seas.upenn.edu

Anaheed Ayoub
University of Pennsylvania
anaheed@seas.upenn.edu

Radoslav Ivanov
University of Pennsylvania
rivanov@seas.upenn.edu

Oleg Sokolsky
University of Pennsylvania
sokolsky@cis.upenn.edu

Insup Lee
University of Pennsylvania
lee@cis.upenn.edu

ABSTRACT

Fault diagnosis in networked systems has been an extensively studied field in systems engineering. Fault diagnosis generally includes the tasks of fault detection and isolation, and optionally recovery (FDIR). In this paper we further consider the blame assignment problem: given a system trace on which a system failure occurred and an identified set of faulty components, determine which subsets of faulty components are the *culprits* for the system failure.

We provide formal definitions of the notion *culprits* and the *blame assignment* problem, under the assumptions that only one system trace is given and the system cannot be rerun. We show that the problem is equivalent to deciding the unsatisfiability of a set of logical constraints on component behaviors, and present the transformation from a blame assignment instance into an instance of unsatisfiability checking. We also apply the approach to a case study in the medical device interoperability scenario that has motivated our work.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring; D.2.1 [**Requirements/Specifications**]: Methodologies—*blame assignment*; D.2.4 [**Software/Program Verification**]: Formal methods

Keywords

Blame Assignment; Component-based System; Trace Analysis; Fault Diagnosis

1. INTRODUCTION

A central idea in systems engineering is that complex systems are built by assembling components. Component-based systems are desirable because they allow independent development of system components by different suppliers, as well as their incremental construction and modification. The down side of component-based development is that no single entity – neither the integrator, nor component suppliers – have a complete understanding of component behaviors and possible interactions between them. This incomplete knowledge, in turn, requires us to resort to black-box analysis methods, when only the input-output behavior of a component is specified.

In this work, we are interested in the forensic analysis of a system following the discovered violation of system safety properties. While this problem is common to all safety-critical domains, our immediate motivation comes from the domain of medical devices. In the United States, the Food and Drug Administration (FDA) is responsible for assessing safety of medical devices and regulating their use in health care. When a system failure that harms a patient, known as an *adverse event* occurs, the hospital is required to report it to the FDA-maintained database [8]. Diagnosis of the root cause is crucial for the subsequent recovery and follow-up prevention measures. Such diagnosis requires recording of system executions leading to the failure, as well as methods for the efficient analysis of the recorded data.

There has been a great amount of research following the seminal work of [5] and [16] in the study of fault diagnosis. In this paper, we take a step further and consider the problem of blame assignment for component-based systems. The system model we use is described in Section 4.1. In this framework, the correct system behavior is captured by a *system contract*. The correct behavior for each component is captured in turn by a *component contract*.

We assume that no information is available about the internal behavior while only the values into and out of a component can be observed. This is especially true in most medical devices used currently, which has motivated collaborative efforts to bring open-source devices to hospitals [2]. We further assume that only one trace is observed. The technique of rerunning the system under varying conditions to observe more traces has been used in some of the existing diagnosis approaches (e.g., [16, 5, 20]). However, rerunning may not always be feasible in practice given the time, financial cost or the potential impact on patients [14, 3].

We make an initial attempt in defining the *blame assignment problem* with the aforementioned assumptions. Intuitively, blame assignment is one step further than fault diagnosis: assuming the faulty components in the system are identified, the blame assignment problem aims to determine the subsets of the faulty components that are responsible for the observed system failure.

We propose a formal solution to the blame assignment problem. Given only the observed trace, we provide for-

HiCoNS'13, April 9–11, 2013, Philadelphia, Pennsylvania, USA.

malized reasoning rules to determine how the system would behave if a suspected set of components are replaced with correct ones. This yields a reconstructed set of all potential system traces that could happen. On each reconstructed trace in the set, we check whether the system violation disappeared, and make a decision of whether to blame a subset of suspected components accordingly. In this paper, we focus on a conservative policy of casting blame, i.e., we require the system violation to disappear on *every* reconstructed trace so as to blame a subset of faulty nodes. In such a case, we call the subset a *culprit*. Based on our formulation, different blame assignment policies can be easily expressed and are discussed in Section 2.

Further investigation of the relations between the identified culprits yields a minimal culprit set. Soundly identifying minimal culprits can make system maintenance cost-effective, as it is enough to only replace components in the minimal culprits instead of all faulty components.

The proposed approach is illustrated with a running example described in Section 4. This example is adapted from a typical medical device interoperability scenario [1, 13].

The paper is organized as follows. We first define the blame assignment problem in Section 2 and discuss related work in Section 3. Then we show the system models used in our approach in Section 4. Our approach is then discussed in Section 5 and Section 6. Finally we conclude with our current work on tool implementation for blame assignment (Section 7) and a discussion in Section 8. A case study based on a medical device interoperability scenario is used throughout the paper to illustrate the approach.

2. THE BLAME ASSIGNMENT PROBLEM

Given an observed trace Tr for system S on which a system property φ_S is violated (denoted $Tr \not\models \varphi_S$), define

$$\mathcal{F} = \{A \mid A \text{ is a component in } S \text{ and } Tr \not\models \varphi_A\} \quad (1)$$

to be the set of faulty components for the violation of φ_S, where φ_A is the constraint on component A's behavior. Not all faulty components in $\mathcal{F}$ necessarily contribute to the system violation. For example, some components may be faulty but unrelated with a certain system property.

In this paper we focus on identifying the subsets of faulty components $\mathcal{F}$ which are the culprits of the violation of system property φ_S. To formally define the notion of a culprit set, we consider a suspected subset $\mathcal{A} \subseteq \mathcal{F}$ of faulty components. Replacing every component in $\mathcal{A}$ with a correct one would result an alternative system S'. Let

$$TR_{\mathcal{A}} = \{tr \mid tr \text{ is a trace for } S' \text{ and } tr \text{ has the same system input as observed on } Tr\} \quad (2)$$

be the set of possible system traces for S' when rerunning the system S' with the same system input. A formal characterization of $TR_{\mathcal{A}}$ is given after Proposition 2 of Section 6. Intuitively, $\mathcal{A}$ is a contributory cause[17] for the system property violation if for *some* $tr \in TR_{\mathcal{A}}$, $tr \models \varphi_S$, i.e., the violation of system property φ_S disappears on *some* system traces after replacing the components in $\mathcal{A}$ with correct ones.

Let $Corr(\mathcal{A}) = \{tr \mid tr \in TR_{\mathcal{A}} \text{ and } tr \models \varphi_S\}$. We define a ratio $p : 2^{\mathcal{F}} \to [0, 1]$ as

$$p(\mathcal{A}) = \frac{|Corr(\mathcal{A})|}{|TR_{\mathcal{A}}|}. \quad (3)$$

Proposition 1. *A subset $\mathcal{A} \subseteq \mathcal{F}$ of faulty components is a contributory cause for the violation of a system property φ_S on an observed trace Tr if and only if $p(\mathcal{A}) > 0$.*

For the computation of the value of $p(\mathcal{A})$, explicit enumeration of the sets $TR_{\mathcal{A}}$ and $Corr(\mathcal{A})$ can be a choice for small-sized, discrete event systems[4, 19]. As the system complexity increases, the size of the set $TR_{\mathcal{A}}$ could grow intractable. Instead of computing the value for p, sampling in the potential trace space to obtain an estimate $\hat{p}$ of p could be used. Similar techniques have already been used in the work of fault diagnosis [5, 20].

The ratio p can be viewed as a measure of likelihood for a subset $\mathcal{A}$ of faulty components to be the contributory cause. The larger the value, the more likely the components in $\mathcal{A}$ contributed to the system property violation.

When assigning blame to components causing system property violation, the ratio p can also reflect different blame assignment policies. If a policy postulates that at least one subset $\mathcal{A}_1$ must be blamed, one may choose the subset which maximizes the ratio p:

$$\mathcal{A}_1 = \arg\max_{\mathcal{A} \in 2^{\mathcal{F}}} p(\mathcal{A}). \quad (4)$$

The calculated or estimated p ratio can also be compared to an empirical value $p_{blame} \in [0, 1]$, where the blame is assigned to a subset $\mathcal{A}$ only when $p(\mathcal{A}) > p_{blame}$. An aggressive policy would choose a small value of p_{blame} (e.g., 0 for the most aggressive policy) and blame a subset $\mathcal{A}$ whenever $p(\mathcal{A}) > p_{blame}$. A conservative policy may choose a much larger threshold p_{blame} in the range $[0, 1]$.

In the wide spectrum from aggressive policies to conservative ones, in this paper, we are interested in the special case where $p(\mathcal{A}) = 1$. It represents the most conservative blame assignment policy with the highest confidence level. Such a subset $\mathcal{A}$ of faulty components represent the main contributory cause[17] for the observed system property violation. By using the approach we introduce in this paper, we are able to cast blame on such a subset $\mathcal{A}$ with the highest confidence level.

In cases where for two subsets $\mathcal{A}$ and $\mathcal{A}'$ such that $p(\mathcal{A}') = p(\mathcal{A})$ and $\mathcal{A}'$ is a proper subset of $\mathcal{A}$, $\mathcal{A}'$ is blamed instead of $\mathcal{A}$. This also indicates a conservative policy where, blaming less is preferred to blaming more. The behaviors of the faulty, but non-blamed, components are not deemed as the main contributory cause for the observed system failure.

We say that a subset $\mathcal{A}$ is minimal at a value p_c if $p(\mathcal{A}) = p_c$ and there does not exist a subset $\mathcal{A}'$ such that $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{A}' \neq \mathcal{A}$, and $p(\mathcal{A}') = p_c$. We define a subset $\mathcal{A}$ to be a *culprit* if $p(\mathcal{A}) = 1$. Given a system S and a trace Tr such that $Tr \not\models \varphi_S$, let $\mathcal{F}$ be as defined in (1), then the *blame assignment* problem is to identify the set

$$Culprit = \{\mathcal{A} \in 2^{\mathcal{F}} \mid \mathcal{A} \text{ is minimal at } 1\}. \quad (5)$$

3. RELATED WORK

Comparison with fault diagnosis. Fault diagnosis [5, 16, 7, 4, 19] in component-based, discrete event systems has been studied in various work in different settings. In these works, a *fault* is defined as "a physical condition that causes a device, a component or, an element to fail to perform in a required manner[4]." Faults may cause *errors* in individual components, which may then propagate through components and lead to system *failures*. Fault diagnosis gener-

ally includes the tasks of fault detection and isolation, and optionally recovery (FDIR). Fault detection aims at determining whether the components contain faults; fault isolation aims at determining the type and location of the faults; whereas fault recovery aims at providing steering feedbacks or remedy actions to the system.

Different from the task of fault isolation, the blame assignment problem is to assign blame to components responsible for the system property violation with the highest confidence level. By assuming that fault detection on an observed system trace has been performed and its result is an input to our approach, we focus on identifying the minimal culprits for system property violations. Therefore, one difference of our work from fault isolation is that, the identified culprit sets can be proper subsets of $\mathcal{F}$. This indicates a scenario where not all faulty components should be blamed for the system property violation: some faulty components's behaviors may have been caused by others producing the wrong input, or some may be irrelevant to the system property, though being faulty. In the worst case, our approach gives the set $\mathcal{F}$ as the culprit, meaning that the faulty behaviors of all components in $\mathcal{F}$ are the main contributory cause for the system property violation.

Use of the ratio p. In this paper, we define the problem using the ratio p that we showed in Section 2. In addition to blame assignment, several existing techniques [4, 20] on fault diagnosis implicitly used such a ratio.

Comparison with the work in [10]. The work of Gössler et al. [10] has been a precursor to ours in attempting to establish the necessary and sufficient causal relationships between components' behaviors and system failure. Our work differs from [10] in the notion of causes and the rules by which traces are reconstructed. The culprits we define in this paper are in fact contributory causes [17], a characterization of an informal reasoning process. For trace construction, the approach in [10] requires to only change the faulty components' behaviors while keeping non-faulty components' behaviors (input and output) unchanged. This ignores the impact of changing one component's behavior on other components and imposes unnecessary constraints on trace reconstruction. The example in Subsection 8.5 illustrates the undesired limitation from this rule.

In addition, in this paper we provide a complete formalization of using state-of-the-art SAT/SMT solvers for efficient computation of culprits, which are not discussed in [10]. On the other hand, the work in [10] uses state machines as component models, which may lend their approach advantages in expressivity of component behaviors.

Comparison with work of higher order contracts. The work in [9] and [11] in the field of programming languages study should not be confused with ours, despite the same terms used such as contracts and blame. The contracts in their line of work refer to the correct type check of parameters in a function call, and blame is cast on either the caller or the callee. In contrast, the blame assignment problem we try to solve is for component-based systems.

4. SYSTEM MODEL AND TRACES

In this section, we present the language we use to define a component-based system. We first informally introduce a simple system that we will use as a running example throughout the paper. We then present the formal definitions of a system in Section 4.1, and illustrate the definitions using our example system in Section 4.2.

The system S in Figure 1 consists of three *components* C, L, and V. Each component has named *input ports* and *output ports*, which are typed variables (a through h). The components are connected by two *channels*, one from port c to port d and the other from port e to port f. Unconnected component ports become the ports for the system S (a, b, g, and h).

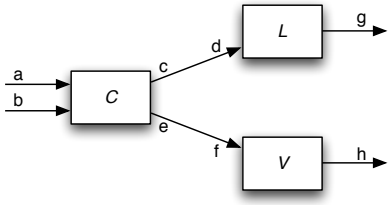

Figure 1: Example System

The behavior of each component is specified as a logical constraint on its input/output pairs of values, called its *contract*. For example, for L, if the the input port d and output port g are of type boolean, then the contract $\varphi_L := g = d$ requires that L produces the same output as the input.

A system contract is similarly defined on system input ports and output ports. We also refer to a system contract as a *system property*.

A *trace* for the system is a map from all ports to their respective observed values. For example, $\{a \to 98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to T, h \to T\}$ is a trace for the system S in Figure 1.

4.1 Formal Definitions

A port $x : \mathbb{T}_x$ is a typed variable with name x and type $\mathbb{T}_x$.

A component A is a tuple $\langle I_A, O_A, \varphi_A \rangle$ where

- $I_A = \{i_1, \ldots, i_m\}$ is a set of input ports,
- $O_A = \{o_1, \ldots, o_n\}$ is a set of output ports, and
- φ_A, called the *contract* for the component, is a logical constraint on ports in $I_A \cup O_A$.

A system $S = \langle A_1, \ldots, A_J, \theta, \varphi_S \rangle$ is a set of components $A_1, \ldots, A_J$ connected by a set θ of *channels*, with a system property φ_S.

A channel for S is a pair of ports (x, y) such that $\mathbb{T}_x = \mathbb{T}_y$, $x \in \bigcup_{j=1}^{J} O_{A_j}$, and $y \in \bigcup_{j=1}^{J} I_{A_j}$. x is called the source of the channel, and y is called the target of the channel. Intuitively, a channel (x, y) for S connects from the output port x of one component of S to the input port y of another component of S.

The system property φ_S for S is a logical constraint defined on unconnected ports of S. Formally, let $I_\theta = \{x \mid \exists y.(x, y) \in \theta\}$ and $O_\theta = \{y \mid \exists x.(x, y) \in \theta\}$, respectively; then, φ_S is a logical constraint defined on ports in $I \cup O$, where $I = \bigcup_{j=1}^{J} I_{A_j} \setminus O_\theta$, and $O = \bigcup_{j=1}^{J} O_{A_j} \setminus I_\theta$ are the input and output ports of S, respectively.

In this paper we make the following assumptions about system construction. (a) Fan-in connections are not allowed, i.e., it is required that $\forall (x_1, y_1), (x_2, y_2) \in \theta . y_1 \neq y_2$. (b) The connected system is acyclic. (c) Each component produces one output on each of its output ports in response to inputs on any subset of its input ports. (d) There is no

name clash among port names. (e) Channels are reliable. A value passed into a channel will be successfully transmitted out to the connected component. In other words, we only consider traces Tr such that $Tr(x) = Tr(y)$ if (x, y) is a channel. Note that assumption (d) can be removed by qualifying port names with the associated component names, as is common in component-oriented languages.

We take a synchronous view of the system execution, where a set of external events arriving at the input ports of the system elicit outputs by the receiving components, which propagate along the connections within the system until outputs are produced on the output ports of the system, and then the system waits until the next set of external events arrive. Given the assumptions (a)–(c) above, each port will be used in any such reaction once. We can thus record the whole response in a single snapshot as values observed at each port. In this work, we consider contracts that describe component behavior in a single snapshot[1]; Section 8.2 discusses an extension to temporal contracts that involve multiple snapshots. Thus, to simplify the discussion, we consider a trace to contain a single snapshot, and formally define it as a map from a port name to the value observed at the port within the reaction.

The value at port x on trace Tr is denoted $Tr(x)$. The expression $Tr(x) = Tr(y)$ states a fact that the value at port x and port y are the same on trace Tr. Given a trace Tr, a component A is said to be *faulty* on Tr if $Tr \not\models \varphi_A$. A system S is said to have violated the property φ_S on the trace Tr if $Tr \not\models \varphi_S$.

Lastly we note that the language for component/system constraints description does not have to be limited to propositional logic. The approach can be generalized to logics whose satisfiability relation is decidable and the number of models for any formula is finite.

4.2 Medical Device Interoperability Case Study

The system S shown in Figure 1 is adapted from a medical device interoperability scenario described in [1, 13].

L is a controller which enables and disables a *laser scalpel* that a surgeon uses to perform chest operation on a patient. V is a controller for the patient ventilator to help keep the oxygen supply of the patient during whole body analgesia.

The system must ensure two properties: (a) the ventilator V is turned on whenever the patient's SpO2 (blood saturation with oxygen) level is below the threshold, and (b) the laser scalpel and the ventilator should not be both turned on.

For property (b), if L and V are both on, then the contact of laser with high concentration of oxygen could cause burn or fire, which is a hazardous situation that must be avoided.

To ensure that violations to properties (a) or (b) do not happen, a *coordinator* component C is used. The coordinator C reads (a) the patient's SpO2 level and (b) the threshold value for SpO2, from ports a and b respectively. If $a < b$, then it sends F to the laser scalpel in port c and T to the ventilator in port e, indicating that the laser scalpel should be disabled and the ventilator should be on; otherwise it sends T to c and F to e.

[1]A snapshot is similar to the notion of measurement in the literature on diagnosis [16], albeit constrained to one reaction.

The laser scalpel and ventilator controller components L and V forward the received instructions in ports d and f to the actual devices via ports g and h, respectively.

Formal definition of the example in Figure 1. The ports a and b are of type integers in between 0 and 100. Other ports are booleans. The formula for checking port value range is

$$\varphi_R(a, b) := (0 \leq a \leq 100) \wedge (0 \leq b \leq 100).$$

The system is defined as $S := \langle C, L, V, \theta, \varphi_S \rangle$, where $\theta = \{(c, d), (e, f)\}$, and

$$\varphi_S := \varphi_R(a, b) \wedge [(a < b) \wedge (\neg g \wedge h) \vee ((a \geq b) \wedge \neg(g \wedge h)]. \quad (6)$$

The component C is defined as $C := \langle I_C, O_C, \varphi_C \rangle$, where $I_C := \{a, b\}$, $O_C := \{c, e\}$, and

$$\varphi_C := \varphi_R(a, b) \wedge [((a < b) \wedge \neg c \wedge e) \vee ((a \geq b) \wedge c \wedge \neg e)]. \quad (7)$$

The component L is defined as $L := \langle I_L, O_L, \varphi_L \rangle$, where $I_L := \{d\}$, $O_L := \{g\}$, and

$$\varphi_L := d = g. \quad (8)$$

The component V is defined as $V := \langle I_V, O_V, \varphi_V \rangle$, where $I_V := \{f\}$, $O_V := \{h\}$, and

$$\varphi_V := f = h. \quad (9)$$

Finally, the constraint on the channels is defined as

$$\eta := (c = d) \wedge (e = f). \quad (10)$$

5. BLAME ASSIGNMENT

In this paper, we focus on blame assignment rather than fault isolation. In blame assignment, we start with the set $\mathcal{F}$ of faulty components for the system property violation on a given trace Tr, and analyze which subsets of $\mathcal{F}$ are the culprits for the system property violation. The set $\mathcal{F}$ is determined by the violations of component contracts observed within Tr.

The reasoning process for identifying culprits is called *trace reconstruction* in this paper. That is, if we suspect a subset $\mathcal{A} \subseteq \mathcal{F}$ to be the culprit, then we analyze how the system would behave should the faulty components in $\mathcal{A}$ be replaced with good ones as specified in their respective contracts. Due to the one-trace assumption that the system will not be rerun to obtain more traces for analysis, we perform this analysis based on information from the component contracts. Given an input to a specific component A, A's output will either be the correct values specified in A's contract, or the faulty value as observed on the trace Tr, depending on the trace reconstruction rules introduced in this section.

By following the approach, a set $TR_{\mathcal{A}}$ of potential system traces will be constructed. We use the definition of the culprit given in Section 2 to determine whether $\mathcal{A}$ is a culprit or not.

The result of blame assignment is usually a subset of $\mathcal{F}$ which caused the system property violation, whereas not all faulty components are blamed. To illustrate this, consider the trace $Tr = \{a \to 98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to T, h \to T\}$ for the system shown in Figure 1.

By the contracts of the components and the system, it is straightforward to decide that, on trace Tr, both C and L are faulty, and a violation to the system property φ_S occurs.

In this case, it can be seen that either replacing the component L or the component C can prevent the system violation of the system property. It is not required that both of them be replace.

In the rest of the section, we use the above example and trace Tr to illustrate a straightforward process of trace reconstruction and analysis for culprits. In the next section, we show how the blame assignment problem can be solved efficiently.

For the system shown in Figure 1 and trace $Tr = \{a \to 98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to T, h \to T\}$, components C and L are faulty and the system property φ_S is violated. We would like to determine which subsets of $\mathcal{F} = \{C, L\}$ are the culprits for the violation of φ_S, without additional reruns of the system to obtain new traces.

An informal analysis for the component L would proceed as: "the system property violation could have been prevented if component L were behaving correctly by giving $\{g \to F\}$ as the output." This analysis uses the implicit assumption that, even if L is replaced with a good one, the faulty component C would misbehave in the same manner as observed, given the same input for component C.

An informal analysis for the component C would proceed as: "the system property violation could have been prevented if component C were behaving correctly by giving $\{c \to T, e \to F\}$ as the output." Here, two implicit assumptions are used. First, when a component is replaced with a good one, its output would change to the correct values accordingly. Second, good components will keep correct behaviors in reconstructed traces, so V produces $\{h \to F\}$ once its input has been changed to $\{f \to F\}$.

Such implicit assumptions as in the above two examples are the best one can assume about components' behaviors, given the one-trace assumption in our problem definition.

As a first step in providing a formal approach to blame assignment, in this paper we explicitly state the informal assumptions as reasoning rules when one has to answer the question: "How would the system behave if a component is replaced with a good one, assuming that the same input is given to the new system?"

For a component A and a suspected subset $\mathcal{A} \subseteq \mathcal{F}$ of faulty components, the *traces reconstruction rules* are as follows.

(R1) If $A \notin \mathcal{F}$, then it is deemed as a good component. In the trace reconstruction, if its input is the same, then its output is kept the same as observed.

(R2) If $A \notin \mathcal{F}$ but its input has changed to other values than observed, then each of the correct output values corresponding to the changed input should be considered as a possible execution the component could perform.

(R3) If $A \in \mathcal{A} \subseteq \mathcal{F}$, then A is a faulty component that is replaced by a good one. Its behavior is the same as a good component in Rule (R2).

(R4) If $A \in \mathcal{F} \setminus \mathcal{A}$, i.e., A is faulty but not in the consideration of being suspected, then no matter whether its input has changed or not, its output remains the same as the value on the observed trace.

The rationale for Rule (R4) is that, for a faulty component which we do not replace due to the one trace assumption, we know no information on how it is supposed to behave other than as observed on the only available trace. Thus it is assumed that it will keep producing the same faulty value.

An example of using the rules is as follows. Consider the system in Figure 1, the trace $Tr = \{a \to 98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to T, h \to T\}$, and an analysis session that only L is suspected (though C and L are both faulty on this trace).

We first take the same input $\{a \to 98, b \to 95\}$ as observed on the trace Tr. Then we consider the component behavior changes according to the topological order of their dataflow.

Since C is faulty but not suspected, the behavior of C is kept as observed on Tr, i.e. C keeps producing the values $\{c \to F, e \to T\}$ (Rule (R4)). Component V is not faulty, so it will behave correctly as specified in (9) to produce $\{h \to T\}$ (Rule (R1)). Component L is suspected, so it will be replaced and behave as a correct one, producing $\{g \to F\}$ as the output (Rule (R3)).

Therefore, the potential system trace after component L is replaced is the reconstructed trace $Tr' = \{a \to 98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to F, h \to T\}$. Thus we obtained a set of possible system behaviors after replacing L, i.e., $TR_{\{L\}} = \{Tr'\}$.[2] Since on Tr' the system property is not violated, we have $Corr(\{L\}) = \{Tr'\}$, thus $p(\{L\}) = 1$. By definition in (5), $\{L\}$ is a culprit.

By similar reasoning with trace reconstruction rules (R1)–(R4), we have $p(\{C\}) = 1$ and $p(\{C, L\}) = 1$, so they are culprits for the system violation as well. The subsets $\{L\}$ and $\{C\}$ are all minimal at ratio 1, therefore $Culprit = \{\{L\}, \{C\}\}$.

In general, given a system $S = \langle A_1, \ldots, A_J, \theta, \varphi_S \rangle$ and a trace Tr such that $Tr \not\models \varphi_S$, the straightforward blame assignment process is as follows.

1. Compute the set $\mathcal{F}$ of faulty components.
2. Let $\mathcal{C}$ be an empty set.
3. For each non-empty subset $\mathcal{A} \subseteq \mathcal{F}$:
   - 3.1 Let $Corr(\mathcal{A})$ be an empty set.
   - 3.2 Use trace construction rules (R1)–(R4) to obtain the set $TR_{\mathcal{A}}$ of reconstructed system traces.
   - 3.3 Examine each trace Tr' in $TR_{\mathcal{A}}$, determine if $Tr' \models \varphi_S$. If yes, put Tr' into $Corr(\mathcal{A})$.
   - 3.4 Compute $p(\mathcal{A})$ according to the definition in Equation (3). If $p(\mathcal{A}) = 1$, put $\mathcal{A}$ into $\mathcal{C}$.
4. Compute $Culprit$ according to the definition in Equation (5), using the collected culprit sets in $\mathcal{C}$.
5. Output $Culprit$.

However, this straightforward computation is time consuming. Indeed, as shown in Theorem 1 in Section 6, an instance of the problem of determining whether $\mathcal{A}$ is a culprit is equivalent to an instance of an unsatisfiability checking problem in the logic used to express component contracts, whereas the latter is known to be a problem in the coNP-complete complexity class even for boolean logic[18].

On the other hand, also due to the equivalence shown in Theorem 1, we could transform the culprit determination problem into an equivalent unsatisfiability checking problem, for which standard best-effort solutions exist. State-of-the-art SAT/SMT solvers can be employed for efficient implementations to find culprits. This transformation is discussed in the next section.

[2]The example shown here is a special case, which only has one reconstructed trace. If a component allows non-deterministic outputs for an input, then a set of traces would be constructed when the component is non-faulty or suspected.

6. TRANSFORMATION INTO AN UNSATISFIABILITY CHECKING PROBLEM

The translation from deciding if $\mathcal{A}$ is a culprit to an unsatisfiability checking problem is made possible by the fact that both the conditions and the corresponding constraints on component behaviors in reconstruction rules (R1)–(R4) can be encoded in logical constraints analogous to Equations (6)–(9).

In this section, we present the transformation, assuming a given system $S = \langle A_1, \ldots, A_J, \theta, \varphi_S \rangle$, and a given trace Tr with $Tr \not\models \varphi_S$.

We first construct a series of subformulas used in later definitions (see Section 6.1 for details):

$$\iota, \eta, \tag{11}$$

$$\xi_{A_j,k}, \quad \text{for } (1 \leq j \leq J,\ k = 1,2,3,4), \text{ and} \tag{12}$$

$$\kappa_{A_j}, \quad \text{for } (1 \leq j \leq J). \tag{13}$$

Here, ι is a formula constraining the input to the system be the same as observed on trace Tr. η is a formula constraining that the two values on the source and target ports of any channel are the same. $\xi_{A_j,k}$ represents the condition check of Rule (Rk) for component A_j. κ_{A_j} represents the constraint on component A_j's behavior if it is supposed to keep the output as on the observed trace in the trace reconstruction, i.e., if Rule (R1) or (R4) applies. The corresponding behavior of component A_j if it is non-faulty, or faulty and suspected (thus replaced with a correct one) is the same as its contract φ_{A_j}. Note that these definitions are parametric to the set $\mathcal{F}$ of faulty components and the set of suspected components $\mathcal{A} \subseteq \mathcal{F}$.

With the defined subformulas, it is then possible to define the behavior of a component in the trace reconstruction:

$$\psi_{A_j} := [(\xi_{A_j,1} \vee \xi_{A_j,4}) \wedge \kappa_{A_j}] \vee [(\xi_{A_j,2} \vee \xi_{A_j,3}) \wedge \varphi_{A_j}]. \tag{14}$$

Informally, this means, if Rule (R1) or (R4) applies, then the component A_j's behavior is constrained by κ_{A_j}; otherwise it is constrained by its contract φ_{A_j}.

Proposition 2. *The formula*

$$\psi := \iota \wedge \eta \wedge \psi_{A_1} \wedge \cdots \wedge \psi_{A_J} \tag{15}$$

defines the set $TR_{\mathcal{A}}$ of all the possible system behaviors with the same input as observed on Tr, after suspected components are replaced with correct ones.

Formally, $TR_{\mathcal{A}} = \{tr \mid tr \models \psi\}$ where ψ is defined in Equation (15) above.

According to the definition of $p(\mathcal{A})$ in Equation (3), for $p(\mathcal{A})$ to be 1, we are left to check that on every trace in $TR_{\mathcal{A}}$ the property φ_S is satisfied. This is equivalent to checking that

$$\psi \wedge \neg\varphi_S \tag{16}$$

is unsatisfiable.

Theorem 1. *Given a system $S = \langle A_1, \ldots, A_J, \theta, \varphi_S \rangle$ with components $A_1, \ldots, A_J$ and a system trace Tr such that $Tr \not\models \varphi_S$. Let $\mathcal{F}$ be defined as in* (1) *and let $\mathcal{A} \in 2^{\mathcal{F}}$. Then $\mathcal{A}$ is a culprit if and only if the formula $\psi \wedge \neg\varphi_S$ defined in Equation* (16) *is unsatisfiable.*

Proof. First suppose (16) is unsatisfiable. Then for any trace tr, either (a) $tr \not\models \psi$ or (b) $tr \models \psi$ but $tr \not\models \neg\varphi_S$. This means that if trace $tr \in TR_{\mathcal{A}}$, then it must be that $tr \not\models \neg\varphi_S$, i.e., $tr \models \varphi_S$, i.e., $tr \in Corr(\mathcal{A})$. Therefore $p(\mathcal{A}) = 1$, so $\mathcal{A}$ is a culprit.

Conversely, suppose $\mathcal{A}$ is a culprit, then $\forall tr \in TR_{\mathcal{A}}.tr \models \varphi_S$, so $tr \not\models \neg\varphi_S$, hence $tr \not\models \psi \wedge \neg\varphi_S$. On the other hand, for trace $tr \notin TR_{\mathcal{A}}$, by the definition of $TR_{\mathcal{A}}$, $tr \not\models \psi$, hence $tr \not\models \psi \wedge \neg\varphi_S$. Therefore, for any trace tr, Equation (16) is not satisfied, i.e., (16) is unsatisfiable. □

6.1 Formula Construction

We now expand the definitions in Equations (11)–(13). The construction is systematic for any system S and trace Tr as defined in Section 4.1, and parametric to the set $\mathcal{F}$ of faulty components and a non-empty subset $\mathcal{A} \subseteq \mathcal{F}$ of suspected components.

A constraint that a port x should have the same as observed on trace Tr is written as a logical formula

$$same(x) := x = Tr_x, \tag{17}$$

where Tr_x is the value of x as observed on trace Tr.

The constraint ι on system input is defined as

$$\iota := \bigwedge_{x \in I} same(x), \tag{18}$$

where I is the set of open input ports of system S.

The constraint η on channels is

$$\eta := \bigwedge_{(x,y) \in \theta} x = y. \tag{19}$$

The test of whether a component A_j is in a set $\mathcal{A}$ is a logical disjunction:

$$in(A_j, \mathcal{A}) := \bigvee_{A \in \mathcal{A}} A = A_j. \tag{20}$$

For a component A_j, the conditions for Rules (R1)–(R4) can then be defined respectively as follows.

$$\xi_{A_j,1} := \neg in(A_j, \mathcal{F}) \wedge \bigwedge_{x \in I_{A_j}} same(x). \tag{21}$$

$$\xi_{A_j,2} := \neg in(A_j, \mathcal{F}) \wedge \neg \bigwedge_{x \in I_{A_j}} same(x). \tag{22}$$

$$\xi_{A_j,3} := in(A_j, \mathcal{A}). \tag{23}$$

$$\xi_{A_j,4} := in(A_j, \mathcal{F}) \wedge \neg in(A_j, \mathcal{A}). \tag{24}$$

The constraints κ_{A_j} on the components output if their output values should be the same as observed is defined as:

$$\kappa_{A_j} := \bigwedge_{x \in O_{A_j}} same(x). \tag{25}$$

For each subset $\mathcal{A}$, using the formulas defined in (17)–(25) to replace those used in (14)–(16), we obtain an instance of the unsatisfiability problem. By courtesy of Theorem 1, we can check for the unsatisfiability of the constructed formula instead of explicitly constructing the set $TR_{\mathcal{A}}$ and checking system property φ_S on every trace in $TR_{\mathcal{A}}$.

6.2 Case Study Continued

Currently we are working on employing the presented approach to perform analyses on the laser scalpel/ventilator interoperability case study, shown in Section 4.2. As an illustration, we show the case for the system trace $Tr = \{a \rightarrow$

$98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to T, h \to T\}$ and suspected set $\mathcal{A} = \{L\}$ which is a subset of $\mathcal{F} = \{C, L\}$ of faulty components.

The logical formulas for the case are constructed according to Equations (18) through (25). For instance,

$$\begin{aligned} \iota &:= (a = 98) \wedge (b = 95), \\ \eta &:= (c = d) \wedge (e = f). \end{aligned}$$

For component C, $\xi_{C,1}$ and $\xi_{C,2}$ are defined to be

$$\begin{aligned} \xi_{C,1} &:= \neg in(C, \mathcal{F}) \wedge (a = 92) \wedge (b = 95), \\ \xi_{C,2} &:= \neg in(C, \mathcal{F}) \wedge \neg[(a = 92) \wedge (b = 95)], \end{aligned}$$

while the definitions for $\xi_{C,3}$, $\xi_{C,4}$, and κ_C are the same as Equations (23), (24) and (25), respectively. Note that the formula

$$\kappa_C := (c = T) \wedge (e = F). \tag{26}$$

represents the case that the faulty component C keeps producing the same wrong values, according to Rule (R4).

The constructions for L are

$$\begin{aligned} \xi_{L,1} &:= \neg in(L, \mathcal{F}) \wedge (d = T) \wedge (g = F), \\ \xi_{L,2} &:= \neg in(L, \mathcal{F}) \wedge \neg[(d = T) \wedge (g = F)], \end{aligned}$$

with $\xi_{L,3}$, $\xi_{L,4}$, and κ_L the same as in Equations (23), (24) and (25), respectively. This is similar for V.

Lastly, an instance of the formula $\psi \wedge \neg\varphi_S$ in Equation (16) is defined. This formula is *unsatisfiable*, which, by Theorem 1, means that the set $\{L\}$ is a culprit. By similar processes, the sets $\{C\}$ and $\{C, L\}$ are culprits. By Definition 5, we have $Culprit = \{\{L\}, \{C\}\}$.

This example shows the difference between blame assignment and fault isolation. In fault isolation, after finding out that L and C are both faulty, the task is to find out what types of faults components L and C may have encountered and where the faults may be located inside the components L and C, respectively. In blame assignment, the task is to identify the minimal subsets of components which have contributed to the system property violation. In the above example, although the components L and C are both faulty, the blame assignment analysis we have performed identifies L and C as individual culprits for the violation, whereas the combination that the violations of both L and C occur is not necessary for the system property violation.

7. IMPLEMENTATION

The construction shown in Section 6.2 is a general process, given the system definition and a recorded trace. Existing state-of-the-art SAT/SMT solvers, such as Z3 from Microsoft Research[6] used in this paper, support the encoding of formula objects definitions. With the formulas encoded, a call to the theorem prover to check whether the formula in Equation (16) is satisfiable is issued, and the result is parsed for determining a culprit.

To automate this process, we have implemented a Python utility that employs the Z3 theorem prover to identify the $Culprit$ set defined in (5). The system description and trace are written in two separate XML files and fed to the utility. Component contracts φ_{A_j} and system property φ_S are written as strings that will be parsed into Z3 formula objects.

After reading in and parsing the XML files, the utility (a) computes the set $\mathcal{F}$, (b) constructs the set $2^{\mathcal{F}}$, (c) for each (non-empty) $\mathcal{A} \in 2^{\mathcal{F}}$, constructs corresponding Z3 formula objects used in (16), (d) calls the Z3 library for unsatisfiability check, (e) records the suspected set if it is a culprit, and finally (f) computes minimal culprits and outputs all the culprits it gathered.

8. DISCUSSION

8.1 Relationship Between Component and System Contracts

We assumed in the paper that component and system contracts are logically related to reflect the correct system design. Formally, for a system $S = \langle A_1, \ldots, A_J, \theta, \varphi_S \rangle$, we require that

$$\bigwedge_{j=1}^{J} \varphi_{A_j} \wedge \eta \to \varphi_S, \tag{27}$$

where η is the port constraint defined in Equation (19). That is, by composing components to design the system S, it should be made sure that the system property is not violated for any accepted input. If this requirement does not hold, the analysis in our approach would not proceed as the set $\mathcal{F}$ of faulty components can be spurious.

This can be illustrated using a slight variation of the case study shown in Subsection 4.2. Everything else being the same, suppose the contract for C had mistakenly included the assignment $\{a \to 98, b \to 95, c \to F, e \to T\}$ as a correct input/output pair. Then on the trace $Tr = \{a \to 98, b \to 95, c \to F, d \to F, e \to T, f \to T, g \to T, h \to T\}$ in the running case study, C would not even be identified as a faulty component in the first place. The set $\mathcal{F}$ of faulty components in this case is just $\{L\}$, whereas the blame assignment procedure introduced in our paper gives $Culprit = \{L\}$ only.

This spurious result is due to the violation of the condition in Equation (27). The given trace Tr in this case is an assignment for the port variables that falsifies (27).

8.2 Extending to Temporal Contracts

For the clarity of presentation, in this paper we have treated a trace as just one snapshot. In general, a system engages in repeated interactions with its environment, and contracts for both the system and individual components can describe relationships between values produced over multiple reactions.

The first complication in handling this general case is that interactions may overlap. For example, if a system is a chain of components, the last component in the chain may still be producing a system output for one reaction, while the first component may have already consumed the next system input, starting the next reaction. Isolating a set of values that belong in a snapshot is in itself a challenging problem and has been studied under the name of trace alignment [12].

Assuming such alignment is possible, we can represent the trace as a sequence of snapshots. We can then use a specification (for example, using a temporal logic) to describe relationships between values in different snapshots. For our case study, an example of a temporal property may be that whenever an SpO2 reading (port a) is less than the threshold (port b), then the output in port g of the laser scalpel component L must be F three snapshots later. We express this property using the "next snapshot" operator X of the linear temporal logic [15] as $a < b \Rightarrow XXX(g = F)$. In this case,

a violation will be detected with a delay of three snapshots, once we observe the laser scalpel output. To perform the blame assignment analysis, we can start trace resonstruction three steps in the past from the moment a violation is observed.

In general, however, it is difficult to tell how far in the past the trace reconstruction should extend. In [10], trace reconstruction starts at the origin of the trace; however the complexity of trace reconstruction grows with the length of the trace. Incremental trace reconstruction may be one possibility to explore.

8.3 Dealing with Timed Systems

In this paper contracts of the system/components are specified as relations on input/output pairs. The analysis does not apply to cases where a system failure would consist of timing information of the values in component ports.

In general, modeling time in the proposed approach can be challenging. One has to provide abstractions to represent time in the system. In addition to considering different values a port could produce, one has to consider different times at which the value could be produced. This additional, orthogonal dimension of complication could dramatically increase the space of potential system traces after the replacement of components. The trace reconstruction idea presented in the paper has to be extended to cope with the time domain. This is currently one aspect of our ongoing work.

8.4 Scalability

Two aspects affect the scalability of our approach. First, in order to investigate each possible combination of faulty components, we explicitly constructed the power set $2^{\mathcal{F}}$ for the set $\mathcal{F}$ of faulty components for the observed system trace. The scalability of our approach is limited by the number of faulty components, rather than the total number of the components in the system.

Second, unsatisfiability problems for boolean logic are known to be coNP-complete[18], which imposes an algorithmic upper bound on analyzer capabilities. However, we envision that the scalability of our approach is only limited by the state-of-the-art SAT/SMT solvers being used. This is due to the fact that our transformation of blame assignment problem instances into unsatisfiability checking problem instances only imposes minimal overhead: as seen in Section 6, the number of variables for constructing logical formulas is the same as the total number of different ports of components, and the number of constructed clauses is linear to the number of components in the system.

We are now working with case studies of larger sizes to obtain empirical results on the above two aspects of limitations on the scalability of our approach.

8.5 Comparison of Reasoning Rules for Trace Reconstruction

In this subsection, we present a slightly revised version of the laser scalpel and ventilator interoperability case study which demonstrates the difference between our approach and the work presented in [10].

In summary, the approach used in [10] to trace reconstruction requires that all non-faulty components' behaviors be kept unchanged in the reconstruction. This effectively rules out traces where the input to non-faulty components has been changed. To accommodate the loss of reconstructed traces, the approach in [10] uses a different reasoning rule for assigning blame, that is, as we interpret, the suspected local component is blamed if and only if this test succeeds: in every possible reconstructed trace where non-faulty components' behaviors are kept unchanged, if the violation on the suspected local component disappears, then the system property violation must also disappear.

This trace reconstruction and reasoning rule is equivalent to ours with only one exception, which happens when there are good components lying downstream in the topological order according to data flow. The omission of certain traces in this approach could lead to spurious analysis results as we illustrate using the system S_1 in Figure 2. In addition to the system in Figure 1, two components L_1 and V_1 are added, whose behaviors are analogous to those of L and V, i.e., forwarding the received messages. However, the system property is now changed to

$$\varphi_{S_1} := \varphi_R(a,b) \wedge [((a < b) \wedge \neg k) \vee ((a \geq b) \wedge k)], \quad (28)$$

i.e., the output l of the lower branch of the system is not related to system property at all. (Note: This is a contrived system property, but it does illustrate the point.)

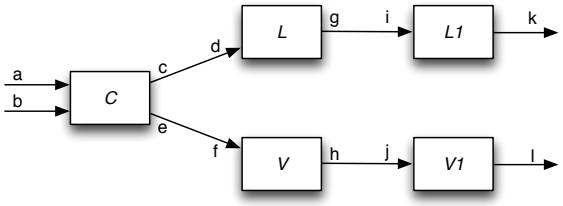

Figure 2: Modified Interoperability Case Study

In this case, on the trace $Tr = \{a \to 98, b \to 95, c \to T, d \to T, e \to F, f \to F, g \to F, h \to T, i \to F, j \to T, k \to F, l \to T\}$, the components L and V are faulty, and the global system property φ_{S_1} is violated.

When analyzing whether the subset $\{V\}$ is a culprit, the trace reconstruction using the approach in [10] is limited by the output of C and the input of V_1 on the observed trace, where $\{e \to F, j \to T\}$. This means on any possible reconstructed trace, it must be that $\{f \to F, h \to T\}$, where V's violation does not disappear. Thus, the precondition of the test rule in [10] for culprit is vacuously false, which makes the test for culprit vacuously succeed. Therefore V is blamed in this case.

This result is apparently spurious, since the system property φ_{S_1} has nothing to do with the component V (and V_1) by construction. In this case, an analysis engine should not ever cast blame on any subset of $\{V, V_1\}$.

As a comparison, our approach considers what the resulting traces would be should the faulty components be replaced with good ones. According to our analysis, the subsets $\{L\}$ and $\{L, V\}$ are culprits, while $Culprit = \{\{L\}\}$.

9. CONCLUSION AND FUTURE WORK

In this paper, we have proposed the blame assignment problem under the assumption that only one system trace is available. We showed that our presented approach casts blame on faulty components for the system property violation in a conservative yet high-confident manner.

The problem that we defined is equivalent to an unsatisfiability checking problem, for which state-of-the-art theorem provers exist. The theorem provers can be utilized for an efficient blame assignment engine implementation. Based on our Python utility which automates the processes of translating blame assignment instances and analyzing for culprits, larger-sized case studies will be studied for us to gain empirical results on the scaleability of our approach.

10. ACKNOWLEDGEMENTS

The research is supported in part by the National Science Foundation grants CNS-0834524, CNS-0930647, and CNS-1035715. We would like to thank FDA researchers Paul L. Jones and Yi Zhang for their motivating discussions on the problem of blame assignment. We would also like to thank Gregor Gössler for the in-depth discussion on their paper [10] and challenges in the blame assignment problem.

11. REFERENCES

[1] The MDPnP website. www.mdpnp.org.

[2] The OSMD website. http://osmdmadison.wordpress.com.

[3] Australian Transport Safety Bureau. In-flight upset–Airbus A330-303, VH-QPA, 154 km West of Learmonth, WA, 7 October 2008. Technical report, Australian Transport Safety Bureau, 2011.

[4] S. Bhattacharyya, Z. Huang, V. Chandra, and R. Kumar. A discrete event systems approach to network fault management: detection and diagnosis of faults. In *Proceedings of the American Control Conference*, volume 6, pages 5108–5113, 2004.

[5] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[6] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS'08*. Springer, 2008.

[7] A. Dubey, G. Karsai, R. Kereskenyi, and N. Mahadevan. Towards a real-time component framework for software health management. Technical Report ISIS-09-111, Vanderbilt University, 2009.

[8] FDA. FDA MAUDE Database.

[9] R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *International Conference on Functional Programming*, ICFP '02, pages 48–59, New York, NY, USA, 2002. ACM.

[10] G. Gössler, D. L. Métayer, and J.-B. Raclet. Causality analysis in contract violation. In *Proceedings of the First international conference on Runtime verification*, RV'10, pages 270–284, 2010.

[11] M. Greenberg, B. C. Pierce, and S. Weirich. Contracts made manifest. *JFP*, 22(3):225–274, 2012.

[12] R. Jagadeesh Chandra Bose and W. van der Aalst. Trace alignment in process mining: opportunities for process diagnostics. *Business Process Management*, pages 227–242, 2010.

[13] C. Kim, M. Sun, S. Mohan, H. Yun, L. Sha, and T. F. Abdelzaher. A framework for the safe interoperability of medical devices in the presence of network failures. In *ICCPS'10*, pages 149–158, 2010.

[14] NERC Steering Group. Technical analysis of the August 14, 2003, blackout: What happened, why, and what did we learn? Technical report, North American Electric Reliability Council, 2004.

[15] A. Pnueli. The temporal logic of programs. In *Proceedings of FOCS '77*, pages 46–57, 1977.

[16] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[17] R. Riegelman et al. Contributory cause: unnecessary and insufficient. *Postgrad Med*, 66(2):177, 1979.

[18] S. Rudich and A. Wigderson. *Computational complexity theory*. American Mathematical Soc., 2004.

[19] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[20] A. Zeller. Isolating cause-effect chains from computer programs. In *ACM International Symposium on Foundations of Software Engineering*, pages 1–10, 2002.