Automated Trust Analysis of Copland Specifications for Layered Attestations[*]

Paul D. Rowe
The MITRE Corporation
USA
prowe@mitre.org

John D. Ramsdell
The MITRE Corporation
USA
ramsdell@mitre.org

Ian D. Kretz
The MITRE Corporation
USA
ikretz@mitre.org

ABSTRACT

In distributed systems, trust decisions are often based on remote attestations in which evidence is gathered about the integrity of subcomponents. Layered attestations leverage hierarchical dependencies among the subcomponents to bolster the trustworthiness of evidence. Copland is a declarative, domain-specific language for specifying complex layered attestations. How phrases are composed bears directly on the trustworthiness of the evidence they produce, and complex phrases become quite difficult to analyze by hand. We introduce an automated method for analyzing executions of attestations specified by Copland phrases in an adversarial setting. We develop a general theory of executions with adversarial corruption and repair events. Our approach is to enrich the Copland semantics according to this theory. Using the model finder Chase, we characterize all executions consistent with a set of initial assumptions. From this set of models, an analyst can discover all ways an active adversary can corrupt subcomponents without being detected by the attestation. These efforts afford trust policymakers the ability to compare attestations expressed as Copland phrases against trust policy in a way that encompasses both static and runtime concerns.

KEYWORDS

Layered Attestation, Model Finding, Trustworthy Computing

ACM Reference Format:
Paul D. Rowe, John D. Ramsdell, and Ian D. Kretz. 2021. Automated Trust Analysis of Copland Specifications for Layered Attestations. In *Proceedings of the 23rd Symposium on Principles and Practice of Declarative Programming, PPDP 2021, Tallinn, Estonia, September 6–8, 2021.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/NNNNNNN.NNNNNNN

1 INTRODUCTION

Network-based communication among computing devices increasingly relies on a notion of trust to inform the nature of their interactions. Remote attestation is a technique for allowing one entity (the target of attestation) to provide evidence of its trustworthiness to a peer (the appraiser of an attestation). It consists, in part, of processes on the target system that gather evidence by performing integrity measurements of various components of the target system. The evidence generated about these components is bundled together and transmitted to a remote peer who can appraise the evidence. The result of the appraisal can form an input to a trust decision that will govern how the network interaction will proceed. For example, a remote attestation may serve to provide a corporate VPN gateway with sufficient evidence that a machine wishing to join the network is free of malware and that it conforms to the corporate configuration before access is granted.

Remote attestation faces the following interesting dilemma. If the appraiser is distrustful of the target to begin with, why should the appraiser be any more trustful of the target to faithfully gather and report accurate evidence? The answer typically lies in the layered dependencies among components of the target system.

To illustrate what we mean, consider a simple hypothetical attestation scenario. Alice is logged into her bank's website and attempts to initiate a high-value transfer of money out of her account. The nature of this transaction prompts the bank to ask Alice to confirm her credentials, perhaps via two-factor authentication. But it could also prompt the bank to initiate an attestation of Alice's system so that it can trust there is no malware present (perhaps in the form of a malicious browser extension) that might have hijacked the session to initiate transactions on Alice's behalf. Of course Alice herself would be willing to reveal some extra information about the current state of her system due to the sensitive nature of the transaction. One approach the bank might take is to develop its own browser extension that can list the other extensions present and compare against a whitelist of approved extensions, refusing the transaction if it sees an extension it does not know or trust. However, an adversary able to install a malicious extension may be just as capable of disabling or corrupting the bank's extension. It would be more trustworthy to inspect the browser from outside the browser itself with some special-purpose application.

This may prompt some readers to become skeptical of the trustworthiness of this special-purpose application. For similar reasons, most of the research on remote attestation has focused on how to build trust from the ground up starting in hardware. Hardware is considered significantly harder to compromise than software, so it can provide a strong root of trust for performing and storing integrity measurements. The development of the Trusted Platform Module [9] and Intel's Software Guard Extensions (SGX) [10] represent prominent products in this area. However, hardware is also less flexible than software. This has led to hardware-based attestation solutions that focus primarily on boot-time integrity measurements at the expense of runtime integrity.

Virtualization technologies such as Xen [1] or Microsoft's Hyper-V [25] are good examples of approaches that can provide strong support for software-based runtime integrity measurement. They offer a place for measurers to stand that is better protected than the environment of the target they are measuring. They contribute to

[*]Approved for Public Release; Distribution Unlimited. Public Release Case Number 21-1504. This technical data was developed using contract funds under Basic Contract No. W56KGU-18-D-0004. The view, opinions, and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

PPDP '21, September 6–8, 2021, Tallinn, Estonia
2021. ACM ISBN 978-1-4503-8689-0…$15.00
https://doi.org/10.1145/NNNNNNN.NNNNNNN

a layered architecture in which more privileged functions benefit from the protection offered by lower layers of the system. Such architectures can combine the benefits of hardware roots of trust with the flexibility afforded by software mechanisms. This suggests an approach to the bank scenario described above in which trust is built from the ground up, starting with hardware and moving up through firmware, hypervisor, and kernel to support the top-level inspection of the browser extensions.

Such an attestation would offer strong evidence of the integrity of the browser extensions, but it may appear to some as a lot of work to generate integrity evidence for a simple property. On the other hand, the assurance gained by pushing trust all the way down to hardware could be well worth it for a more high-stakes property. So it is desirable to support the ability to adjust the level of trust required and tailor it to the situation.

Indeed, an attestation can be strengthened in layered architectures even without going all the way down to hardware. Previous work has established a model and a set of reasoning principles for the analysis of layered attestations focused on runtime measurements [21]. These principles support the notion that the trustworthiness of the evidence produced by a layered attestation depends on the order in which evidence is gathered on the target system. This is due to the layered dependencies among components on the target system. In particular, building up trust in components in a "bottom-up" manner is generally more trustworthy than other orders. If an adversary is to avoid being detected, it must either corrupt deeper (and presumably better-protected) components of the target system, or else corrupt components in opportune time windows during the attestation itself. This result is summarized by saying that bottom-up measurements force the adversary to perform corruptions that are either "recent or deep".

But it may not be necessary to force an adversary all the way down to the hardware level. In other words, the components we consider sufficiently "deep" will depend (among other things) on the context of the transaction. The primary targets of measurement for an attestation will also vary depending on the nature of the trust decision being supported by the attestation. What a bank wants to know about a system before approving a large transaction is likely different from what a corporate gateway wants to know about the same system when admitting it onto the corporate network. This suggests that the appraiser and the target will need a way to negotiate details of an attestation such as which components get measured and in what order, how deep the measurements go, and how the evidence is bundled for appraisal.

Copland [19] is a declarative specification language designed to tailor specifications to different target device architectures and different contexts for trust decisions. Its formal semantics enables semantically clear negotiations between a target and a relying party about the details of the attestation to be performed. The flexibility allowed by Copland specifications is crucial for its ability to accommodate the full range of situations in which layered attestations might be performed. However, this flexibility underscores the importance of understanding the trust properties provided by alternative specifications. The "bottom-up" rule is insufficient in part because it does not say how deep is deep enough. An appraiser should be able to determine what deeper attestations buy them so they can consider the trade-offs between trust and other factors. For instance, all else being equal, a speedier option may be preferred to one that takes more time to complete.

Our Contributions. In this paper, we build primarily off of three prior efforts ([18, 19, 21]) combining them in a novel way. We explicitly apply the reasoning principles developed in [21] to the trust analysis of Copland phrases as introduced in [19]. Since even moderately large Copland phrases can be prohibitively difficult to analyze by hand, we introduce a method for automating the reasoning principles. The basic goal of our analysis is to determine all the essentially different ways an adversary can corrupt a given subcomponent while avoiding detection by a layered attestation designed to attest to the integrity of the subcomponent. Our approach is to use Chase [18], a general-purpose model finder for geometric logic, to enumerate all the possible executions consistent with the Copland specification in which the adversary corrupts the target and avoids detection. This set of executions contains all the information needed to understand the conditions under which each subcomponent must be trusted.

More concretely, our novel contributions in this paper are as follows:

(1) We develop a first-order theory of *saturated queries* in Section 4.2 that adapts the reasoning principles of [21] to the analysis problem of finding all models that violate the attestation goals. We identify a correctness criterion for our theory and prove that our theory meets that criterion (Thm. 12).
(2) Since Chase is a model finder that works on the geometric fragment of first-order logic, in Section 6 we find a suitable axiomatization of our theory in special geometric form and prove the equivalence of the geometric theory to the more naturally-stated first-order theory developed in Section 4.2 (Thm. 16). This two-stage axiomatization ensures we accurately capture the reasoning principles laid out in [21] while allowing us to leverage Chase for automation.
(3) We demonstrate the use of an end-to-end tool chain in Section 7 that compiles a Copland phrase into its event semantics [19] and then performs the trust analysis yielding a characterization of all possible ways for an adversary to defeat the attestation. From this characterization the analyst can determine the least amount of work an adversary must perform to defeat the attestation. We also show how an analyst can alter assumptions about dependency relationships on the target and about adversary capabilities. This allows the analyst to tailor the analysis according to their needs.

The remainder of the paper is structured as follows. In Section 2, we introduce a typical attestation scenario to introduce the context and some notation, and to motivate our goals. Section 3 provides a high-level overview of the syntax and semantics of Copland from [19]. Section 4 reviews the reasoning principles of [21] and adapts them into the first-order theory of saturated queries. In Section 5, we briefly introduce our Chase model finder [18] and how it works. Section 6 provides the translation of the theory from Section 4 into special geometric form. Section 7 walks the reader through several examples of using Chase to analyze the trust properties of Copland phrases under a variety of assumptions. After presenting some related work, we finally conclude.

2 MOTIVATION

In this section, we use the client-bank scenario to illustrate the utility of the Copland language. A full explanation of Copland syntax and semantics follows in the next section.

The client must provide evidence to the bank that its web browser is free of malicious extensions before a transaction can proceed. A simple version of this attestation may take place among the bank, a browser monitor bmon running in the client's userspace us, and a host-based antivirus suite av running in the client's kernelspace ks. The target of the attestation is exts, the client's browser extensions. The bank can choose to have the client use either or both of av and bmon to generate evidence. As we will see, the bank's choices greatly influence the trustworthiness of the evidence it receives.

We envision an active adversary able to interfere with the proper function of components on the client. This adversary may *corrupt* a component via any intervention that causes it to deviate (to the adversary's advantage) from its *regular* behavior. The adversary may also *repair* corrupted components, returning them to their regular states. The corruption state of a component has two important implications for measurement: a regular measurer will always accurately report the corruption state of its target, and a corrupted measurer will always report a regular state for its target.

Though powerful, this adversary is not omnipotent. It cannot tamper with evidence after the fact, nor can it prevent measurements from occurring. The adversary can however delay measurements while respecting any orderings chosen by the bank.

We consider corruptions in narrow timeframes or of better-protected components to be difficult for this adversary. We refer to these respectively as *recent* and *deep* corruptions. On the other hand, we consider repairs in any timeframe and corruptions of less-protected components without time constraints to be easier for this adversary.

When a component is corrupted but evidence gathered during an attestation passes appraisal, we say the adversary has *avoided detection* at the component. A critical question is: what other components must the adversary have corrupted to avoid detection at a particular target? This is a central concern we must address to understand which measurement topologies most constrain adversarial behavior, and therefore which we should choose.

The Copland phrase in Example 1 represents one choice the bank may make for structuring the client's attestation. It encompasses two measurements taken in parallel, indicated by the ∼ operator joining them. The bank issues separate measurement requests to av and bmon and receives the evidence. Here, "parallel" means that the measurement ordering is not specified: any order is acceptable.

Example 1.

$$*\mathsf{bank} : @_{\mathsf{ks}}\ [\mathsf{av}\ \mathsf{us}\ \mathsf{bmon}]\ {+}{\sim}{+}\ @_{\mathsf{us}}\ [\mathsf{bmon}\ \mathsf{us}\ \mathsf{exts}]$$

In the left measurement specification, av from its place in ks measures bmon in us. On the right, bmon in us measures exts, also in us. We can easily see the motivation to choose these measurements: if the better-protected av reports that bmon is regular, bmon's measurement of exts can be reasonably trusted to be accurate.

One can interpret the Copland phrase in Example 1 as a precise shorthand for a partially-ordered set of labeled measurement events. Every legal Copland phrase has a well-defined formal event semantics. This phrase's semantics includes two events, corresponding to its two measurement specifications, and no orderings. The labels associated with these events are msp(ks.av, us.bmon) and msp(us.bmon, us.exts).

Unfortunately, not constraining the order in which the measurements occur opens this attestation up to easy subversion. Suppose the adversary has corrupted one of the client's browser extensions. The adversary can arrange for bmon to measure before av. They can then corrupt bmon before it measures exts. The corrupted measurer bmon will falsely report that exts is regular. The adversary may then repair bmon before allowing av's measurement to proceed. Because bmon is regular when it is measured, av reports no corruption. Both evidence artifacts pass appraisal, meaning the adversary avoids detection at exts.

How difficult is this to achieve? The adversary had to corrupt bmon, a relatively less-protected component, without time constraints, as well as repair bmon. Both actions are easy for this adversary, meaning this subversion is overall easy to carry out. That it is viable is a consequence of insufficiently constraining measurement precedence. Measurement order is as important to consider as the measurements themselves: the bank should order the measurements more deliberately.

The Copland phrase in Example 2 is identical to that in Example 1 but for one critical distinction: av's measurement of bmon is now required to occur before the latter measures exts. This is indicated by the < operator joining the two measurements.

Example 2.

$$*\mathsf{bank} : @_{\mathsf{ks}}\ [\mathsf{av}\ \mathsf{us}\ \mathsf{bmon}]\ {+}{<}{+}\ @_{\mathsf{us}}\ [\mathsf{bmon}\ \mathsf{us}\ \mathsf{exts}]$$

Here, delaying av's measurement offers no advantage, as this will also delay bmon's measurement. To avoid detection at exts, the adversary now has two choices: corrupt av and bmon before the attestation begins or else corrupt bmon after it is measured by av but before it measures exts. Thus, to avoid detection, the adversary must choose between a deep corruption of av or a recent corruption of bmon, both of which are hard. Prior theoretical work has shown this to be a general feature of bottom-up measurement topologies like this one, in which deeper components measure shallower ones before these perform their own measurements [21]. The bank should choose this topology because it maximally constrains the adversary.

Decisions about which components measure and in what order can have striking implications for the trustworthiness of evidence. For very simple attestations like these, with only a small number of events in the semantics of their Copland phrases, it is easy enough to apply our theory of runtime corruption and work out appraisal outcomes by hand. However, dependency structures and trust relationships in real-world systems can be extremely complex. The sheer number of components one must countenance in these systems would easily overwhelm human analysts. The goal of this paper is to develop a method for automating this analysis and thereby scale it up to meet the challenges of real-world attestation problems.

C	$\leftarrow$	$S\ P\ S$	Measurement (Probe Place Target)
	$\mid$	$@_P\ [\ C\ ]$	At place
	$\mid$	$\{\}$	Nullify
	$\mid$	$\_$	Copy
	$\mid$	$!$	Sign
	$\mid$	#	Hash
	$\mid$	$C\ \to\ C$	Linear sequence
	$\mid$	$C\ D{<}D\ C$	Sequential branching
	$\mid$	$C\ D{\sim}D\ C$	Parallel branching
	$\mid$	$(\ C\ )$	Grouping
D	$\leftarrow$	$-\mid+$	Splitting specification

Figure 1: Copland Syntax

3 COPLAND LAYERED ATTESTATION LANGUAGE

Copland is a declarative language for specifying layered attestations. It is used to declare the location and means by which a component is measured, the methods for combining the evidence collected from each measurement, and the constraints on the sequencing of measurement actions. It has a precise semantics based on partially ordered sets of events [19, Sec. 5]. Possible events include, among others, measurement actions to generate evidence, and hashing and signing to protect previously generated evidence.

Copland assumes that each action is performed at a location called a *place*. The origin of an attestation request p for phrase t is specified with the syntax $*p : t$. All phrases we consider in this paper are of the form $*\text{bank} : t$. The syntax of Copland phrases is presented next.

A symbol S is a sequence of lowercase letters. A place P is a sequence of digits or a symbol. A legal Copland phrase t is in the syntactic category C as defined in Fig. 1. The branching operators are non-associative and have the same precedence, and the linear sequencing operator is right associative and has a higher precedence.

A complete description of the combining operators in Copland requires a description of the flow of evidence collected by a phrase, however, for this paper, the particulars of evidence collection are irrelevant. Instead, we focus on the events used to collect and combine evidence, and their orderings. Features of the language that are relevant only for reasoning about evidence will be identified in the presentation of the language.

Semantically, a Copland phrase specifies the mapping of input evidence to some output evidence via one or more attestation events. Each event occurs at a well defined place. Events involve taking measurements, signing or hashing evidence, or routing evidence so as to produce combinations of evidence.

To describe the semantics of executing Copland phrase C at place P, we describe how it transforms evidence making use of diagrams of the form:

$$\begin{tikzcd} {} \arrow[r] & \boxed{\text{label}} \arrow[r] & {} \end{tikzcd}$$

The box represents an event, and the arrows show the flow of evidence. The ordering of events respects the flow of evidence. The syntax of an event label is given in Figure 2.

$$\begin{array}{rcl} L & \leftarrow & \mathsf{msp}(P.S, P.S) \mid \mathsf{nul}(P) \mid \mathsf{cpy}(P) \mid \mathsf{sig}(P) \mid \mathsf{hsh}(P) \\ & \mid & \mathsf{req}(P,P) \mid \mathsf{rpy}(P,P) \mid \mathsf{split}(P,D,O,D) \mid \mathsf{join}(P,P) \\ O & \leftarrow & < \mid \sim \end{array}$$

Figure 2: Event Label Syntax

$$\begin{tikzcd} & \boxed{e_1} \arrow[r] & \boxed{e_2} \arrow[r] & \boxed{e_3} \arrow[dr] & \\ \boxed{e_0} \arrow[ur] \arrow[dr] & & & & \boxed{e_7} \\ & \boxed{e_4} \arrow[r] & \boxed{e_5} \arrow[r] & \boxed{e_6} \arrow[ur] & \end{tikzcd}$$

$\ell(e_0) = \mathsf{split}(\mathsf{rp}, +, \sim, +)$	$\ell(e_4) = \mathsf{req}(\mathsf{rp}, \mathsf{us})$
$\ell(e_1) = \mathsf{req}(\mathsf{rp}, \mathsf{ks})$	$\ell(e_5) = \mathsf{msp}(\mathsf{us.bmon}, \mathsf{us.exts})$
$\ell(e_2) = \mathsf{msp}(\mathsf{ks.av}, \mathsf{us.bmon})$	$\ell(e_6) = \mathsf{rpy}(\mathsf{rp}, \mathsf{us})$
$\ell(e_3) = \mathsf{rpy}(\mathsf{rp}, \mathsf{ks})$	$\ell(e_7) = \mathsf{join}(\mathsf{rp})$

Figure 3: Event System for Example 1

The most basic Copland phrase is a measurement $m\ q\ t$, for symbols m and t, and place q, where m names the probe, t names the target of the measurement, and q is the place at which the target resides. When at place p, $m\ q\ t$ means that p should receive some evidence, perform m targeting t at q, and then emit the resulting evidence.

$$\begin{tikzcd} {} \arrow[r] & \boxed{\mathsf{msp}(p.m, q.t)} \arrow[r] & {} \end{tikzcd}$$

The semantics of the phrases nullify $\{\}$, copy $\_$, sign $!$, and hash # have the same form as a measurement. When executing at place p, their corresponding event labels are $\mathsf{nul}(p)$, $\mathsf{cpy}(p)$, $\mathsf{sig}(p)$, and $\mathsf{hsh}(p)$.

Let $p : \{c\}$ be the events and their orderings associated with executing phrase c at p. Measurements can be combined in a pipeline fashion using the $\to$ operator. Thus when at p, $c_1 \to c_2$ means

$$\begin{tikzcd} {} \arrow[r] & p : \{c_1\} \arrow[r] & p : \{c_2\} \arrow[r] & {} \end{tikzcd}$$

A measurement can be taken at a remote location using the @ operator. When at p, $@_q\ [c]$ means

$$\begin{tikzcd} {} \arrow[r] & \boxed{\mathsf{req}(p,q)} \arrow[r] & q : \{c\} \arrow[r] & \boxed{\mathsf{rpy}(p,q)} \arrow[r] & {} \end{tikzcd}$$

Phrases c_1 and c_2 can be combined using branching. There are two ways of combining phrases using branching, sequential ($o = <$) and parallel ($o = \sim$) combination. They both follow the same split-join pattern. When at p, $c_1\ d_1 o d_2\ c_2$ means

$$\begin{tikzcd} & & p : \{c_1\} \arrow[dr] & & \\ {} \arrow[r] & \boxed{\mathsf{split}(p, d_1, o, d_2)} \arrow[ur] \arrow[dr] & & \boxed{\mathsf{join}(p)} \arrow[r] & {} \\ & & p : \{c_2\} \arrow[ur] & & \end{tikzcd}$$

The split specifications d_1 and d_2 only effect the flow of evidence. When $d = +$, evidence is passed, and when $d = -$, evidence is dropped. There are additional orderings associated with sequential branching. When $o = <$, all of the events associated with c_1 precede the events associated with c_2.

Formally, the semantics of a Copland phrase is given by an event system.

Definition 3 (Event System [19]). An *event system* $(E, \prec, \ell)$ consists of

(1) set E of events,

(2) relation $\prec \subseteq E \times E$, a strict partial order, and
(3) function $\ell : E \longrightarrow L$, a map from events to labels.

The event system for Example 1 is presented in Figure 3. The branching operation $\sim$ contributes events e_0 and e_7. Measurements occur at events e_2 and e_5. Events e_1 and e_3 cause the measurement at e_2 to occur at ks. Events e_4 and e_6 cause the measurement at e_5 to occur at us. The strict partial order $\prec$ is the transitive closure of the $\longrightarrow$ relation shown in the diagram in Figure 3.

The events and labels for Example 2 are the same as they are for Example 1 except that $\ell(e_0) = \mathsf{split}(\mathsf{rp}, +, <, +)$. The other difference is the nodes are linearly ordered.

$$\begin{tikzcd}
\boxed{e_0} \arrow[r] & \boxed{e_1} \arrow[r] & \boxed{e_2} \arrow[r] & \boxed{e_3} \arrow[dll, bend left] & \\
& \boxed{e_4} \arrow[r] & \boxed{e_5} \arrow[r] & \boxed{e_6} \arrow[r] & \boxed{e_7}
\end{tikzcd}$$

In this work, we will focus solely on measurement events, and abstract away all other kinds of events. As a result, sequential branching and pipelines have the same semantics.

4 THEORY OF LAYERED ATTESTATION

This section concerns understanding how an adversary can interfere with an attestation. Section 4.1 is primarily a summary of the main definitions of [21]. Section 4.2 is entirely new.

4.1 Review of Prior Definitions

Our primary interest is in analyzing the trust of a Copland phrase in the presence of an active adversary. We thus start by enriching the partially ordered event semantics for Copland with labels for two new types of adversary events: corruption of a component, and repair of a component.

$\ell(e) = \mathsf{cor}(p.c)$ asserts that event e corrupts component $p.c$, and
$\ell(e) = \mathsf{rep}(p.c)$ asserts that event e repairs component $p.c$.

Adversary events have no associated evidence.

These adversary events serve to toggle the corruption state of system components between being regular or corrupted. In turn, the corruption state of components will affect the behavior of other events. In the current work, we consider an adversary model in which corrupted components only effect the outcome of measurement events. For all other events (req, rpy, etc.) corrupted components have no effect.

Measurement events produce new pieces of evidence about the target being measured. While the actual values of the evidence data will be complex and varied, our primary concern is whether the evidence passes or fails appraisal at the relying party. We therefore adopt the following idealization of the outcome of measurement. A measurement event e can generate evidence that will either pass or fail appraisal. We also assume that a corrupted measurer has the incentive and ability to always generate evidence that will pass appraisal, regardless of the corruption state of the target of measurement. Conversely, (with one exception described below) a regular measurer always generates evidence whose appraisal accurately describes the corruption state of the target. So if the target is regular at the time of measurement the measurer will produce evidence that passes appraisal, and if the target is corrupt, the measurer will generate evidence that will not pass appraisal.

Table 1: The effect of corruption on measurement.

Measurer	Context	Target	Evidence Appraisal
corrupt	*	*	passes
*	corrupt	*	passes
regular	regular	regular	passes
regular	regular	corrupt	fails

In the latter case we say that the measurement event detects the corruption.

The one exception to the above rule is when the measurer relies on some other component in its context to work effectively. A good example of this is antivirus software. Its access to the file system is mediated through the operating system kernel. So, if the kernel is corrupted, it could hide a corrupted target file. Thus, whenever a measurer has such a contextual dependency, and that component is corrupted, we assume it always has the incentive and ability to cause the measurer to generate evidence that will pass appraisal. These assumptions are summarized in Table 1, where we use * to indicate that the outcome of measurement is the same regardless of the corruption state.

A component may rely on zero, one, or more other components. When $p.m$ depends on $q.c$ we write $\mathsf{Depends}(p.m, q.c)$. Typically the places p and q will be the same so we make that simplifying assumption throughout this paper. Since, according to Table 1, the appraisal only fails when the target is actually corrupt, this means these cases accurately detect corruption. We therefore write $\mathsf{Detects}(e)$ to denote the fact that the measurement event e produces evidence that will fail appraisal, allowing the relying party to detect a corruption. Since we are most interested in cases where the relying party is fooled into trusting a corrupted system, we will focus on instances in which the Detects predicate is empty, i.e. all measurement events generate evidence that will pass appraisal.

Adversary ordered. The assumptions codified in Table 1 only make sense if the corruption state of a component at an event is well defined. To see why there might be an issue, consider the partially ordered set of events in Fig. 4. Since the corruption of $p.m$ is neither before nor after its repair, it is unclear whether we should consider $p.m$ to be corrupt or regular at the measurement event. The problem is that there are two adversary events affecting $p.m$ that are maximal in the precedence ordering before the measurement event. In the adversary-enriched semantics, we need to ensure there is at most one maximal adversary event affecting a given component prior to any event. We only need to worry about maximal corruptions or repairs for components that are *relevant* to the given event.

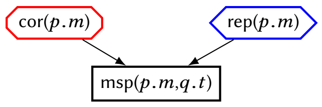

Figure 4: An example where corruption state is ill-defined.

Definition 4 (Relevant [21])**.** Component $p.c$ is *relevant* to event e iff

(1) $\ell(e) = \mathsf{msp}(p.c, q.t)$ or $\ell(e) = \mathsf{msp}(q.m, p.c)$, or
(2) $\ell(e) = \mathsf{msp}(p.m, q.t)$ and $\mathsf{Depends}(p.m, p.c)$, or
(3) $\ell(e) = \mathsf{cor}(p.c)$ or $\ell(e) = \mathsf{rep}(p.c)$.

There is a simple condition that will guarantee the corruption state of components is well defined.

Definition 5 (Adversary-Ordered [21])**.** $(E, \prec, \ell)$ is *adversary-ordered* iff for each $e_1, e_2 \in E$, if $p.c$ is is relevant to both e_1 and e_2, and e_2 is an adversary event, then e_1 and e_2 are comparable events, that is $e_1 \prec e_2$, $e_2 \prec e_1$, or $e_1 = e_2$.

Any adversary-ordered event system E induces a predicate Cor_E that identifies all and only those components that are corrupt at an event to which they are relevant. Specifically, $\mathsf{Cor}_E(p.c, e)$ holds if the (unique) most recent adversary event affecting $p.c$ prior to e is $\mathsf{cor}(p.c)$. If, instead, there is no adversary event affecting $p.c$, or if the most recent such event is $\mathsf{rep}(p.c)$, then $\neg\,\mathsf{Cor}_E(p.c, e)$ holds. Following the rules in Table 1, adversary-ordered event systems also uniquely determine the $\mathsf{Detects}_E$ predicate that identifies which events accurately detect corruptions. To be explicit, $\mathsf{Detects}_E(e)$ iff $\mathsf{Cor}_E(p.t, e)$ holds for the target of measurement $p.t$, and $\neg\,\mathsf{Cor}_E(q.c, e)$ for all other components $q.c$ relevant to e. Otherwise $\neg\,\mathsf{Detects}_E(e)$. For the remainder of the paper, we restrict our attention to adversary-ordered event systems.

The Copland semantics for a phrase t is trivially adversary-ordered. Since there are no corruption events, Cor_E is the empty predicate. However, the purpose of performing an attestation is to detect corruptions of components of interest. We therefore are interested in all the ways to enrich the (adversary-free) Copland event semantics with adversary events.

Definition 6 (Adversary-Enriched)**.** Let $(E, \prec, \ell)$ be an event system. Then $(E', \prec', \ell')$ *adversary-enriches* $(E, \prec, \ell)$ iff

(1) $E \subseteq E'$,
(2) $\prec \subseteq \prec'$,
(3) for $e \in E$, $\ell(e) = \ell'(e)$, and
(4) every $e \in E' \setminus E$, is an adversary event.

We often abuse notation and write E to denote the triple $(E, \prec, \ell)$ leaving $\prec$ and ℓ implicit. Definition 6 defines a natural partial order on executions: $E \leq E'$ iff E' adversary-enriches E.

Definition 7 (Execution)**.** An *execution* of Copland phrase t is an event system that adversary-enriches the Copland semantics for t. We write $\mathcal{E}(t)$ to denote the set of all executions of phrase t.

4.2 Theory of Saturated Queries

The set $\mathcal{E}(t)$ will contain many executions, some of which detect corruptions (i.e. $\mathsf{Detects}_E(e)$ for some measurement event e) and some of which do not. Our aim is to provide a mechanism for querying $\mathcal{E}(t)$ to discover the set of executions that satisfy some assumptions of interest. The most important queries are those that yield executions in which the adversary successfully avoids detection. Thus we may assume that $\mathsf{Detects}_E$ is empty, and that $\mathsf{Cor}_E(p.c, e)$ for some component $p.c$ and some measurement event e. We then want to find all executions in $\mathcal{E}(t)$ consistent with the assumptions.

More generally, we consider queries of the form (E, φ) where φ is a predicate identifying which components we are assuming to be corrupt at which events. That is $\varphi(p.c, e)$ indicates an assumption that $p.c$ is corrupted at event e.

Just as we defined a partial order on event systems above, we can define a partial order on corruption predicates φ by saying $\varphi \leq \varphi'$ iff for all $p.c$ and for all e, $\varphi(p.c, e)$ implies $\varphi'(p.c, e)$. These two orderings combined create a natural partial order on queries (E, φ): $(E, \varphi) \leq (E', \varphi')$ iff $E \leq E'$ and $\varphi \leq \varphi'$. The ordering is strict iff either of the constituent orderings is strict. We use this ordering to formalize our search goal from two paragraphs ago.

Definition 8. The *denotation* of query (E, φ), written $[\![(E, \varphi)]\!]$, is the set of executions defined by the following rules. $E' \in [\![(E, \varphi)]\!]$ iff all of the following hold:

(a) $E \leq E'$
(b) $\varphi \leq \mathsf{Cor}_{E'}$
(c) $\mathsf{Detects}_{E'}$ is the empty predicate.

Condition (a) says we are interested only in adversary enrichments of the given execution E. Condition (b) says that E' satisfies any assumption we made in φ about components being corrupt. Condition (c) says that we are only interested in executions that do not detect corruptions.

Definition 8 is a clear definition of the set we would like to enumerate, but it does not immediately suggest any procedure for doing so. In fact, it suffices to enumerate only those executions in $[\![(E, \varphi)]\!]$ that are minimal in the $\leq$ ordering on event systems. All other executions in the denotation require the adversary to perform at least as much work as one of the minimal ones. In that sense, we don't aim to enumerate all executions in the denotation, but rather, we aim to *characterize* all executions in the denotation by enumerating the minimal ones. The core idea is to perform a search by climbing in the $\leq$ ordering on queries until we reach a stopping condition for a query (E', φ') that allows us to include E' in our enumeration. We next define the stopping condition which relies on the following definition.

Definition 9. The query (E, φ) is *saturated* if and only if

(a) $\varphi = \mathsf{Cor}_E$, and
(b) $\mathsf{Detects}_E$ is the empty predicate.

Condition (a) here is similar to condition (b) in Def. 8, except it further restricts how Cor_E is allowed to extend φ. In particular, it says that E cannot contain any corrupted components at events not already identified by φ. Saturated queries signal that a search in the query ordering can stop, as indicated by the following two lemmas.

Lemma 10. If (E, φ) is saturated, then $E \in [\![(E, \varphi)]\!]$.

We omit the proof as it is a simple application of the definitions.

Lemma 11. If (E, φ) is saturated and $(E, \varphi) < (E', \varphi')$ then $E \notin [\![(E', \varphi')]\!]$.

Proof. Either $E < E'$ or $\varphi < \varphi'$. In the first case, either E' has strictly more events than E or E' has strictly more orderings than E. Either way, this extra structure is preserved by all $\hat{E} \geq E'$, so all elements of $[\![(E', \varphi')]\!]$ must also contain this extra structure. It follows immediately that $E \notin [\![(E', \varphi')]\!]$.

In the other case where $\varphi < \varphi'$, there is some $(p.c, e)$ such that $\neg\varphi(p.c, e)$ and $\varphi'(p.c, e)$. By Def. 8 condition (b), for every element

$\hat{E} \in [\![(E', \varphi')]\!]$, $\mathrm{Cor}_{\hat{E}}(p.c, e)$. But since $\mathrm{Cor}_E = \varphi$ and $\neg\varphi(p.c, e)$, $E \notin [\![(E', \varphi')]\!]$. □

Lemmas 10 and 11 show why the definition of saturated is a useful and natural stopping condition. By Lemma 10, saturated queries identify elements of the denotation, and by Lemma 11 if we didn't stop the search, we would miss the execution encoded by the saturated query. The following theorem tells us that by enumerating the minimal saturated queries above (E, φ), we capture precisely the denotation of (E, φ).

Theorem 12. Let $R = \{(E', \varphi') \mid (E, \varphi) \leq (E', \varphi')$ and (E', φ') is saturated$\}$. Let R_{min} be the $\leq$-minimal members of R. Then

$$[\![(E, \varphi)]\!] = \bigcup_{(E', \varphi') \in R_{min}} [\![(E', \varphi')]\!].$$

Proof. The reverse inclusion is easy to show. We know $E \leq E'$ and $\varphi \leq \varphi'$ by the definition of R_{min}. If $\hat{E} \in \bigcup_{(E',\varphi') \in R_{min}} [\![(E', \varphi')]\!]$, then for some $(E', \varphi') \in R_{min}$, we have $E' \leq \hat{E}$, $\varphi' \leq \mathrm{Cor}_{\hat{E}}$, and $\mathrm{Detects}_{\hat{E}}$ is empty. By the transitivity of $\leq$, we easily conclude that $E \leq \hat{E}$ and $\varphi \leq \mathrm{Cor}_{\hat{E}}$.

Now consider the forward inclusion. Suppose $\hat{E} \in [\![(E, \varphi)]\!]$. Then by Def. 8, $E \leq \hat{E}$, $\varphi \leq \mathrm{Cor}_{\hat{E}}$, and $\mathrm{Detects}_{\hat{E}}$ is empty. Now consider the query $(\hat{E}, \mathrm{Cor}_{\hat{E}})$. It is saturated because, trivially, $\mathrm{Cor}_{\hat{E}} = \mathrm{Cor}_{\hat{E}}$, and we already saw $\mathrm{Detects}_{\hat{E}}$ is empty. Furthermore, we already knew that $\varphi \leq \mathrm{Cor}_{\hat{E}}$. So $(\hat{E}, \mathrm{Cor}_{\hat{E}}) \in R$. Thus there must be some $(E', \varphi') \in R_{min}$ such that $(E', \varphi') \leq (\hat{E}, \mathrm{Cor}_{\hat{E}})$. So $E' \leq \hat{E}$ and $\varphi' \leq \mathrm{Cor}_{\hat{E}}$. So $\hat{E} \in [\![(E', \varphi')]\!]$ as desired. □

Up to now, we have described the theory of saturated queries and showed why that theory correctly captures the denotation we want. We have not produced any search algorithm to enumerate saturated queries. Our approach is to leverage a general-purpose model finder for geometric logic to implement the search. We will provide the model finder with an axiomatization of the theory of saturated queries, and the job of the model finder is to enumerate minimal (and possibly non-minimal) models of the theory. Theorem 12 tells us that if we correctly axiomatize the theory and if the model finder performs correctly, then it will enumerate the set we want. Before turning to our axiomatization of the theory of saturated queries, we first provide an overview of the model finder discussing what it does and how it works.

5 MODEL FINDING WITH CHASE

Chase [18] is a model finder for first-order logic with equality. It finds minimal models of a theory expressed in finitary special geometric form, where functions in models may be partial. A formula is in *finitary special geometric form* if it is a finite sentence consisting of a single implication, the antecedent is a conjunction of atomic formulas, and the consequent is a disjunction. Each disjunct is a possibly existentially quantified conjunction of atomic formulas.

$$\forall \vec{x}.\, \mathsf{P}_1(\vec{x}) \wedge \cdots \wedge \mathsf{P}_n(\vec{x}) \Rightarrow \bigvee_i \exists \vec{y}_i.\, \mathsf{Q}_{i,1}(\vec{x}, \vec{y}_i) \wedge \cdots \wedge \mathsf{Q}_{i,n_i}(\vec{x}, \vec{y}_i)$$

A function is *partial* if it is defined only on a proper subset of its domain. A sentence in first-order logic is *finitary geometric* iff it is logically equivalent to a finite set of sentences in finitary special geometric form. Finitary geometric logic is also called coherent logic.

We will assume familiarity with basic ideas and results from first-order mathematical logic; notions that are not defined here are treated in any text on logic, such as [6], with allowances for partial functions. When a structure A satisfies theory T, we write $A \models T$ and call A a model of T. The definition of a homomorphism must account for partial functions.

Definition 13 (Homomorphism). Let A and B be structures. A *homomorphism* h of A into B is a function with these properties

(1) For each n-place predicate P,
$$\mathsf{P}(a_1, \ldots, a_n) \in A \text{ implies } \mathsf{P}(h(a_1), \ldots, h(a_n)) \in B.$$
(2) For each n-place function f,
$$f(a_1, \ldots, a_n) = a_0 \in A \text{ implies } f(h(a_1), \ldots, h(a_n)) = h(a_0) \in B.$$

We write $A \ll B$ when there is a homomorphism from A into B.

Definition 14 (Minimal Model). Model A of T is *minimal* iff for all models B of T, whenever $B \ll A$ then $A \ll B$.

Definition 15 (Set of Support). A set of models M is a *set of support* for theory T iff whenever $B \models T$, there exists a model $A \in M$ such that $A \ll B$.

When given a theory, Chase produces a set of support whenever it terminates successfully. It may produce some models that are not minimal. We will discuss this point again in Section 7.3.

5.1 Chase Input

The input to the Chase program is a set of first-order formulas in finite geometric form. The official syntax is a slight variation of Geolog [7] syntax, and is inspired by Prolog in the sense that quantification is implicit and variables are distinguished using capitalization, where variables are capitalized. The full details of Chase input syntax is described in [18].

For uniformity of the current presentation, we will display all input formulas in the traditional mathematical style. In particular we use $\top$ and $\bot$ to represent the always true and always false formulas respectively. We also use the convention that logical constants are interpreted as 0-ary function symbols and depicted using sans-serif font: a, b, c, etc. Comments are delimited with (∗ ∗).

As a minimal illustrative example, we can represent the theory of total orders of two or fewer elements with the following input:

(∗ Total Ordering ∗)
$$\top \Rightarrow \mathsf{num}(\mathsf{a}) \wedge \mathsf{num}(\mathsf{b})$$
$$\forall x, y\,.\, \mathsf{num}(x) \wedge \mathsf{num}(y) \Rightarrow x < y \vee x = y \vee y < x$$

where a and b are constants, i.e., 0-ary function symbols.

5.2 Structures

A Chase structure for theory T is a set of facts. A *fact* is a ground atomic formula that has one of two forms

(1) $\mathsf{P}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$, or
(2) $f(\mathsf{c}_1, \ldots, \mathsf{c}_n) = \mathsf{c}_0$.

where P and f are in the signature of T.

The universe of structure A is the least set U of constants such that

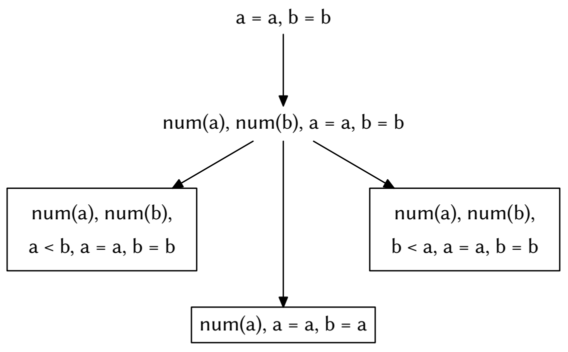

Figure 5: Chase search tree for the total ordering example

(1) $\mathsf{P}(\mathsf{c}_1, \ldots, \mathsf{c}_n) \in A$ implies $\mathsf{c}_1, \ldots, \mathsf{c}_n \in U$, and
(2) $f(\mathsf{c}_1, \ldots, \mathsf{c}_n) = \mathsf{c}_0 \in A$ implies $\mathsf{c}_0, \ldots, \mathsf{c}_n \in U$.

Let C be the set of constants that occur in theory T. A structure A produced by Chase for theory T has the following properties.

(1) Functions may be partial.
(2) Each constant in C is the left hand side of a fact in A.
(3) Equality in A is closed under congruence.
(4) Each element in U is the canonical representative of an equivalence class induced by congruence closure.

One of the three models found by Chase for the total ordering example above is $\{\mathsf{num}(\mathsf{a}), \mathsf{num}(\mathsf{b}), \mathsf{a} < \mathsf{b}, \mathsf{a} = \mathsf{a}, \mathsf{b} = \mathsf{b}\}$.

5.3 Algorithm

Models of theory T are found using an algorithm called the chase [13]. The procedure starts with a structure in which each constant in T is equated to itself. Queue Q is created containing the initial structure, and the main loop begins.

The chase for theory T repeats the following steps until queue Q is empty.

(1) Take structure A from Q.
(2) If A models T then output A.
(3) Otherwise, choose a formula F in T not satisfied by A.
   (a) Find a variable assignment S for the universally quantified variables in F such that its antecedent is satisfied, but its consequent is not.
   (b) Apply S to each disjunct in the consequent.
   (c) For each disjunct, substitute a freshly generated constant for each existentially quantified variable, and add to the queue a structure produced by augmenting A with the disjunct. Mark A as being the parent of the new structure.

The structures generated by a run of Chase on the Total Ordering Example are shown in Fig. 5. An arrow connects a parent structure with a child. The boxed structures are models of the theory.

The initial structure just equates constants. The second structure generated adds num facts. At this point, the only applicable sentence is the one that imposes a total ordering. It produces models, and Chase terminates successfully.

In general, when structure A is the parent of B, there exists a homomorphism from A into B. Every chase step is structure-preserving.

Chase finds models in a similar manner as is done by Razor [23]. The major difference between the two tools is Razor is designed for interactive exploration of models while Chase is designed to work without human intervention. Correctness and termination properties of the chase algorithm are presented in the Razor paper.

6 CHASE THEORY FOR LAYERED ATTESTATION

At the end of Section 4 we identified the goal of correctly axiomatizing the theory of saturated queries. As queries (E, φ) contain adversary-ordered event systems as one component of the pair, our theory contains an axiomatization of adversary-ordered event systems. Since the definitions are already in special geometric form, and due to space limitations, we omit the presentation of this part of the theory to focus on the axiomatization of saturation. We note only that these rules serve to define the meaning of the predicates $\mathsf{relevant}(p.c, e)$ and $\mathsf{ms\_evt}(e)$ which say that component $p.c$ is relevant to event e (in the sense of Def. 4), and that e is a measurement event, respectively.

We also omit numerous rules that do not provide any insight, such as rules that express the injectivity of event labels. While such rules are important for a correct axiomatization, a discussion of them would detract from the core contribution.

We present below a general theory of saturation. Our geometric theory for any given phrase will contain a formula stating that if E is a measurement event, it must be one of the events generated by the Copland semantics. This enforces Def. 6 ensuring models found by Chase are adversary enrichments of the given Copland semantics. That formula is necessarily dependent on the Copland phrase being analyzed, so it is not included as part of the general theory.

Axiomatizing the theory of saturated queries is not as straightforward. The theory as presented in Section 4 relies on the derived predicates $\mathsf{Detects}_E$ and Cor_E. These have properties that are not preserved under model homomorphisms (i.e. adversary-enrichments). That is, if we know $E \leq E'$ we cannot necessarily conclude that $\mathsf{Cor}_E \leq \mathsf{Cor}_{E'}$, or vice versa. Since formulas in finitary special geometric form are precisely those that are preserved under homomorphism, and since the $\leq$ relation on the derived predicates Cor_E and $\mathsf{Detects}_E$ are not preserved under homomorphism, we cannot directly reference $\mathsf{Detects}_E$ or Cor_E in our axiomatization.

We therefore need to find an equivalent set of formulas in finitary special geometric form that do not reference any facts not preserved by homomorphism. We use the four formulas depicted in Fig. 6. Notice that for components $p.c$ the place p and the component c are treated as two separate variables.

Informally, Formula 1 states that if the target of a measurement event is corrupt at the time of measurement, then (under the background assumption that all evidence created will pass appraisal) either the measurer or something it depends on must be corrupt at that event. This essentially encodes Table 1. Formula 2 says that if a component is corrupt at a given event, there must have been a prior corruption event for that component. Formula 3 says that if a component is corrupt at event e_2 but it was previously repaired at e_1, then there must be a corruption event for that component between e_1 and e_2. Finally, Formula 4 says that if there is a corruption event for a given component, and a later measurement event at which the component is relevant, then either the component is still corrupt, or there is an intermediate repair event for the component. Notice that all the formulas rely on existential quantification, and Formulas 1 and 4 use disjunction. We are therefore comfortably outside the Horn fragment, so we cannot rely on tools restricted to Horn clauses.

It is far from obvious that these four formulas are equivalent to the theory of saturated queries. The following theorem demonstrates that they are indeed equivalent.

$$\begin{array}{l}
(*\ \text{Formula 1}\ *)\\
\forall e, p, m, q, t\,.\,\ell(e) = \mathsf{msp}(p.m, q.t) \wedge \varphi(q.t, e)\\
\quad \Rightarrow \varphi(p.m, e) \vee\\
\qquad \exists c\,.\,\mathsf{Depends}(p.m, p.c) \wedge \varphi(p.c, e)
\end{array}$$

$$\begin{array}{l}
(*\ \text{Formula 2}\ *)\\
\forall e, p, c\,.\,\varphi(p.c, e)\\
\quad \Rightarrow \exists e'\,.\,e' \prec e \wedge \ell(e') = \mathsf{cor}(p.c)
\end{array}$$

$$\begin{array}{l}
(*\ \text{Formula 3}\ *)\\
\forall e_1, e_2, p, c\,.\,e_1 \prec e_2 \wedge \varphi(p.c, e_2) \wedge \ell(e_1) = \mathsf{rep}(p.c)\\
\quad \Rightarrow \exists e'\,.\,e_1 \prec e' \wedge e' \prec e_2 \wedge \ell(e') = \mathsf{cor}(p.c)
\end{array}$$

$$\begin{array}{l}
(*\ \text{Formula 4}\ *)\\
\forall e_1, e_2, p, c\,.\,e_1 \prec e_2 \wedge \ell(e_1) = \mathsf{cor}(p.c) \wedge\\
\quad \mathsf{ms\_event}(e_2) \wedge \mathsf{relevant}(p.c, e_2)\\
\quad \Rightarrow \varphi(p.c, e_2) \vee\\
\qquad \exists e'\,.\,e_1 \prec e' \wedge e' \wedge e_2 \wedge \ell(e') = \mathsf{rep}(p.c)
\end{array}$$

Figure 6: The Chase theory for saturated queries.

Theorem 16. *The query (E, φ) is saturated iff it satisfies Formulas 1–4.*

Proof. Keep in mind that if there is no assignment of the universally quantified variables to constants that satisfy the antecedant of a formula, then the formula is trivially satisfied.

($\Rightarrow$): We proceed by contrapositive, assuming one of the four formulas is not satisfied, and conclude that (E, φ) is unsaturated.

Formula 1: Assume Formula 1 is not satisfied. Then there is some measurement event e in which component $p.m$ measures component $q.t$, and $\varphi(q.t, e)$ but $\neg\varphi(p.m, e)$ and for every $p.c$ such that $\mathsf{Depends}(p.m, p.c)$, $\neg\varphi(p.c, e)$. If $\mathrm{Cor}_E \neq \varphi$ then, by definition, (E, φ) is unsaturated which is what we aim to show. So we may assume $\mathrm{Cor}_E = \varphi$. Then $\neg\,\mathrm{Cor}_E(p.m, e)$ and for every $p.c$ such that $\mathsf{Depends}(p.m, p.c)$, $\neg\,\mathrm{Cor}_E(p.c, e)$. Then by the definition of $\mathrm{Detects}_E$ as described in Table 1, $\mathrm{Detects}_E(e)$ and so is not empty. Thus, by definition, (E, φ) is unsaturated.

Formula 2: Assume Formula 2 is not satisfied. Then for some $p.c$ and e, $\varphi(p.c, e)$, but there is no $e' \prec e$ with $\ell(e') = \mathrm{cor}(p.c)$. Then $\neg\,\mathrm{Cor}_E(p.c, e)$, and hence $\mathrm{Cor}_E \neq \varphi$. So, by definition, (E, φ) is unsaturated.

Formula 3: Assume Formula 3 is not satisfied. Then there is some repair event $\ell(e_1) = \mathrm{rep}(p.c)$ with $e_1 \prec e_2$ such that $\varphi(p.c, e_2)$, but there is no intermediate corruption event $e_1 \prec e' \prec e_2$ such that $\ell(e') = \mathrm{cor}(p.c)$. Thus, the last adversary event for $p.c$ before e_2 is a repair event. This means $\neg\,\mathrm{Cor}_E(p.c, e_2)$. But $\varphi(p.c, e_2)$, so $\mathrm{Cor}_E \neq \varphi$, making (E, φ) unsaturated.

Formula 4: Assume Formula 4 is not satisfied. Then there is some corruption event $\ell(e_1) = \mathrm{cor}(p.c)$ preceding a measurement event e_2 to which $p.c$ is relevant where $\neg\varphi(p.c, e_2)$ and there is no intermediate repair event $e_1 \prec e' \prec e_2$ with $\ell(e') = \mathrm{rep}(p.c)$. Since there is no intermediate repair event, the last adversary event for $p.c$ before e_2 is a corruption event. This means that $\mathrm{Cor}_E(p.c, e_2)$. But $\neg\varphi(p.c, e_2)$, so $\mathrm{Cor}_E \neq \varphi$, making (E, φ) unsaturated. This concludes the proof of one direction.

($\Leftarrow$): We suppose that (E, φ) is unsaturated and we demonstrate that one of Formulas 1–4 is not satisfied. We take cases on whether $\mathrm{Cor}_E = \varphi$.

$\mathrm{Cor}_E = \varphi$: In this case, since (E, φ) is unsaturated, there must be an event e for which $\mathrm{Detects}_E(e)$. By examining Table 1, the only possibility is for $\neg\,\mathrm{Cor}_E(p.m, e)$ and for all $p.c$ in the context of $p.m$, $\neg\,\mathrm{Cor}_E(p.c, e)$, whereas the target of measurement $q.t$ satisfies $\mathrm{Cor}_E(q.t, e)$. Since $\mathrm{Cor}_E = \varphi$, $\varphi(q.t, e)$ but $\neg\varphi(p.m, e)$ and $\neg\varphi(p.c, e)$. Thus Formula 1 is not satisfied.

$\mathrm{Cor}_E \neq \varphi$: So there is some event e and some component $p.c$ such that either $\neg\,\mathrm{Cor}_E(p.c, e)$ and $\varphi(p.c, e)$, or $\mathrm{Cor}_E(p.c, e)$ and $\neg\varphi(p.c, e)$.

Suppose that $\neg\,\mathrm{Cor}_E(p.c, e)$ and $\varphi(p.c, e)$. From $\neg\,\mathrm{Cor}_E(p.c, e)$ we know that either there are no adversary events for $p.c$ before e, or, if there are some, the most recent one has the label $\mathrm{rep}(p.c)$. In the first case, Formula 2 is not satisfied, because $\varphi(p.c, e)$ holds without a prior corruption event. In the second case, Formula 3 is not satisfied because $\varphi(p.c, e)$ holds with a prior repair event for $p.c$, but without an intermediate corruption event.

Finally, suppose $\mathrm{Cor}_E(p.c, e)$ and $\neg\varphi(p.c, e)$. From $\mathrm{Cor}_E(p.c, e)$ we know that there is a corruption event for $p.c$ prior to e without any intermediate repair event. But since $\neg\varphi(p.c, e)$, this means Formula 4 is not satisfied. □

Theorem 16 tells us that models found with Chase using the theory of Fig. 6 (together with the theory of adversary-ordered executions) will be saturated queries. As Chase is designed to produce a set of support, if we feed Chase with a formula representing an initial query (E, φ), its output will include all the minimal saturated queries (E', φ') such that $(E, \varphi) \leq (E', \varphi')$. So by Thm. 12 and Lemma 10, we can project these queries onto their executions E' to obtain the minimal elements of the denotation $[\![\,(E, \varphi)\,]\!]$. These minimal executions characterize all the ways an adversary can avoid detection consistent with the original query (E, φ). For any other (non-minimal) execution in the denotation, there is a minimal one in which the adversary performs less work (i.e. strictly fewer actions or strictly weaker orderings). An analyst can then inspect the minimal executions to determine if the work required of an adversary is sufficiently difficult. We demonstrate such analyses in the next section.

7 ANALYSIS OF COPLAND PHRASES

In this section we build on the examples from Section 2 to show how to use Chase and our input theory to analyze the trust properties of Copland phrases.

7.1 Example Walkthrough

We start by returning to Example 1 from Section 2:

$$*\mathsf{bank} : @_{\mathsf{ks}}\ [\mathsf{av}\ \mathsf{us}\ \mathsf{bmon}] +\!\sim\!+ @_{\mathsf{us}}\ [\mathsf{bmon}\ \mathsf{us}\ \mathsf{exts}]$$

This phrase is designed to check the list of extensions exts installed in the browser. In our analysis, therefore, we aim to discover the ways in which the adversary might evade detection assuming exts is corrupt (i.e. contains an unapproved extension) when it is measured. Our initial query, therefore, will be (E, φ) in which E is the event system produced by the Copland semantics, and φ only holds for $\varphi(\mathsf{us.bmon}, \mathsf{e}_5)$ where e_5 is the event in which us. bmon measures us. exts. (The subscript of 5 comes from the labeling in Fig. 3.)

We have implemented a preprocessing pipeline that converts a raw Copland phrase into a Chase formula representing its event system E. This phrase is a logical conjunction of facts that correspond to the existence of events (labeled as measurement events) and facts expressing the precedence order among them. The result when applied to Example 1 is the following.

$$\begin{aligned} \top \Rightarrow\ & \ell(\mathsf{e}_2) = \mathsf{msp}(\mathsf{ks.av}, \mathsf{us.bmon}) \wedge \\ & \ell(\mathsf{e}_5) = \mathsf{msp}(\mathsf{us.bmon}, \mathsf{us.exts}) \end{aligned} \tag{1}$$

The non-consecutive event numbers reflect the presence of non-measurement events in the full Copland semantics described in Section 3. We represent φ explicitly with the following rule.

$$\top \Rightarrow \varphi(\mathsf{us.exts}, \mathsf{e}_5) \tag{2}$$

We then submit to Chase a theory consisting of

- the theory of adversary-ordered executions,
- the formulas in Fig. 6,
- a single formula expressing the fact that any measurement event is one of the ones implied by the Copland semantics,
- Formulas from equations (1) and (2) above representing the initial query, and
- a set of auxiliary formulas expressing lower level facts such as the injectivity of function symbols.

The full set of inputs can be found at copland-lang.org [22].

Chase performs its search as described in Section 5, and Theorem 16 ensures that models found by Chase are saturated queries compatible with the semantics of the given Copland phrase. Example 1 results in 5 different saturated queries. Fig. 7 depicts the executions corresponding to those saturated queries. The fact that the adversary can avoid detection is not surprising because there is always a way for an adversary to succeed by corrupting enough components or by corrupting components in the intervals between when they are measured and when they perform measurements. Indeed two of the models (Models 2 and 3) represent executions in which the adversary utilizes exactly those two strategies. In Model 3, ks.av, us.bmon, and us.exts are all corrupted before the start of the

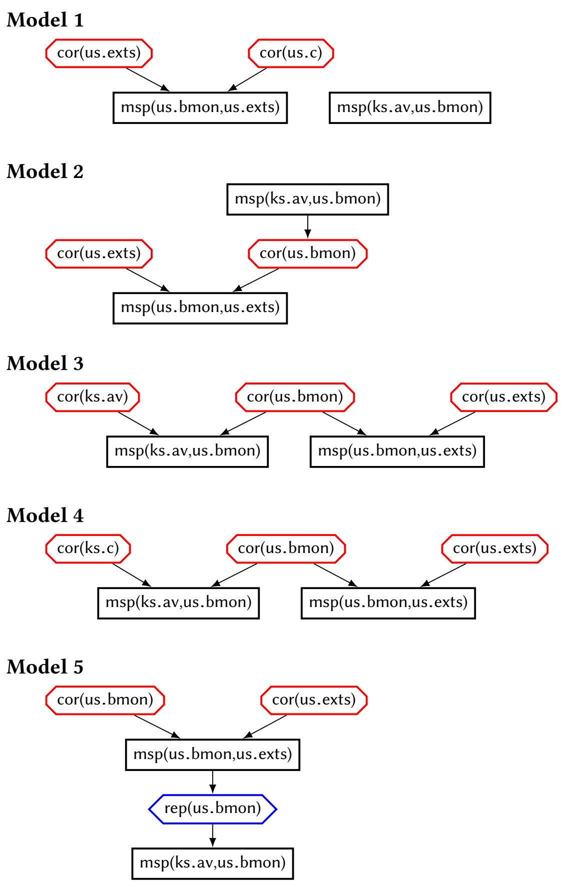

Figure 7: Executions of Example 1

attestation. In Model 2, only us.bmon and us.exts are corrupted, but the former is corrupted *after* it is measured, but *before* it performs its measurement. Two other models (Models 1 and 4) are variants on these in which the measurement components themselves are not corrupted but Chase posits some other component on which they depend might be corrupted instead. Chase cannot identify what such a component might be. It simply recognizes that the existence of such a dependency (which is not itself measured) could lead to the adversary avoiding detection. So, for example, us.av probably relies on the kernel to function correctly. If the kernel is corrupted with a rootkit, this could cause the av not to discover a corruption of us.bmon. Finally, in the fifth model (Model 5), the adversary succeeds by corrupting both us.bmon and us.exts before the attestation begins. But in order to avoid detection at the measurement of us.bmon by ks.av the adversary relies on the possibility that that this measurement could happen *after* the other measurement, providing an opportunity to repair us.bmon after it performs its measurement, but before it gets measured itself.

These 5 models characterize where the remaining risk lies after performing the layered attestation specified by the Copland phrase. They help identify assumptions that must hold in order for the attestation to guarantee detection of corruption. For example, the adversary must not corrupt ks.av or any component it depends on. Similarly, the adversary must not corrupt us.bmon after it has been measured. A relying party may have reason to believe that the target system is capable of ensuring these assumptions are met. For example, the relying party may trust that kernel level protections will protect ks.av from corruption, or that, say, address space layout randomization makes a successful runtime corruption of us.bmon extremely unlikely. Alternatively, the relying party may simply be willing to take these assumptions on blind faith. If the attestation is meant to support an interaction that is not overly sensitive, the relying party might believe the adversary is capable of performing one of these actions, but also be willing to take such a risk.

One advantage of our analysis approach is that we can often explicitly express such assumptions as additional formulas input to the Chase. For example, we can write

$$\forall c \,.\; \mathsf{Depends}(\mathsf{us.bmon}, \mathsf{us.}c) \Rightarrow \bot \tag{3}$$

to express the assumption that the browser monitor does not depend in any way that matters on any other component. If we add this formula to our Chase theory we are essentially asking Chase not to show us any models in which it discovers such a dependency. Indeed, when we run Chase with this extra formula, it identifies 4 models instead of 5 because only one of the original 5 satisfied the antecedent of the formula (and hence was discarded).

Recent or deep corruptions have been identified as winning strategies for an adversary seeking to avoid detection given this adversary model [21]. However, as we saw above, there may be reasons to believe such corruptions are not likely. In either case, it is useful to discover if a Copland phrase *only* admits such strategies, or if there are other ways for the adversary to succeed. We can express the lack of deep corruptions directly by identifying the components which are considered "deep enough" to have strong protection and writing, for example:

$$\forall e \,.\; \ell(e) = \mathsf{cor}(\mathsf{ks.av}) \Rightarrow \bot \tag{4}$$

This excludes models in which ks.av is ever corrupted. We can similarly exclude *all* recent corruptions by disallowing corruption events that occur after some measurement event:

$$\forall e_1, e_2, p, c \,.\; e_1 \prec e_2 \wedge \ell(e_2) = \mathsf{cor}(p.c) \wedge \mathsf{ms\_evt}(e_1) \Rightarrow \bot \tag{5}$$

For the Copland phrase in Example 1, if we add the two formulas (4) and (5) precluding deep or recent corruptions, together with assumptions like equation (3) that express there are no unaccounted-for dependency relationships, then the search finds only one execution. This is Model 5 in Fig. 7 in which a corrupted us.bmon performs its measurement, then repairs itself before getting measured. While this attack may require some skill (or possibly luck) in ensuring the two measurements happen in the required order, a stronger Copland phrase would guarantee the impossibility of this order of measurement. The Copland phrase from Example 2 specifies that ks.av must perform its measurement of us.bmon before the latter performs its measurement. This measurement is "bottom-up" in the sense that components higher up in the layered structure of the target system are measured later than the lower layers. Bottom-up measurement orderings are known to guarantee that any corruptions are recent or deep [21]. It is easy to verify this result for the particular case of Example 2 by submitting a query that excludes recent or deep corruptions. Indeed, Chase finds no executions for Example 2 when recent and deep corruptions are excluded.

7.2 Analyzing Larger Phrases

With Copland phrases as simple as those from Examples 1 and 2, it seems feasible to perform an analysis by hand. The simple examples above allow for a clear exposition of the underlying principles of the methodology. However, as soon we begin to consider more complicated phrases involving numerous components with various dependencies and measurements that can be ordered in many ways, the analysis becomes much more complex and requires automation. As layered attestation is still an emerging technique, it is difficult to say how large a "typical" Copland phrase would be for realistic applications.

Two sources give us an initial indication of how large we might expect them to be. First, our colleagues Adam Petz and Perry Alexander have recently implemented an infrastructure for the faithful execution of Copland phrases [17]. In ongoing work developing a demonstration, they report using Copland phrases that involve 4-5 measurement events [15]. Since this is an initial demonstration, we might expect phrases to grow larger than that. Our second source of information on what might be a typical size of a Copland phrase comes from the Internet Engineering Task Force's Remote ATtestation procedureS (RATS) Working Group. Their document describing an architecture for remote attestation [2] envisions supporting at least three layers (ROM, bootloader, kernel) that collect evidence about the environments above them in the hierarchy. Each layer might have several targets of measurement, easily resulting in 9-12 measurement events. We thus need to ensure that our approach can scale to such sizes and beyond.

As a concrete example, consider the following more complex version of the bank's attestation problem. Instead of using a special-purpose measurer to simply list installed browser extensions, the bank is willing to accept a list generated by the browser's extension manager us.extmgr. However, the bank is interested in gaining trust in the extension manager which relies on part of the core browser code us.bser to function properly. Thus, the bank's special-purpose browser monitor us.bmon would now be responsible for measuring core parts of the browser code as well as the extension manager. For instance, it could hash elements of core functions needed to properly enumerate the list of extensions. The general-purpose antivirus software us.av would still be responsible for scanning for malware affecting us.bmon. For extra assurance, the bank will also request a runtime measurement of the operating system kernel. Tools such as LKIM [12] inspect the structure of the memory of a Linux kernel to detect violations of invariants that commonly occur when rootkits attempt to hide. WinKIM is a similar tool for the Windows kernel, and it could be run in a separate VM supported by Microsoft's Hyper-V virtualization technology. So it may request the kernel integrity monitor in Hyper-V, hv.kim, to measure the kernel, ks.ker. For completeness, we imagine the target system has a way to measure us.av itself from another component hv.avm living in a Hyper-V VM. The following Copland phrase accomplishes the above attestation.

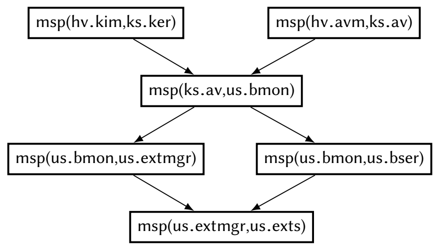

Figure 8: Copland semantics for complex attestation.

Example 17.

$$\begin{aligned}*\mathsf{bank} : @_{\mathsf{hv}}[(&\mathsf{kim\ ks\ ker} +\!\sim\!+ \mathsf{avm\ ks\ av})\\ &+\!<\!+ @_{\mathsf{ks}}[\mathsf{av\ us\ bmon}\\ &\qquad +\!<\!+ @_{\mathsf{us}}[(\mathsf{bmon\ us\ extmgr} +\!\sim\!+\\ &\qquad\qquad \mathsf{bmon\ us\ bser})\\ &\qquad\qquad +\!<\!+ \mathsf{extmgr\ us\ exts})]]]\end{aligned}$$

The value of automation provided by Chase for analyzing such a phrase becomes quickly apparent. In analyzing this phrase, we assume two dependency relations exist. Specifically, we assume both

$$\forall p, c\ .\ \mathsf{Depends}(\mathsf{us.extmgr}, p.c) \Rightarrow p = \mathsf{us} \land c = \mathsf{bser} \qquad (6)$$

$$\forall p, c\ .\ \mathsf{Depends}(\mathsf{ks.av}, p.c) \Rightarrow p = \mathsf{ks} \land\ c = \mathsf{ker} \qquad (7)$$

The Copland semantics for Example 17 is depicted in Fig. 8. If we submit a query to Chase in which we only stipulate that φ holds for us.exts when it is measured assuming only that there are no unaccounted-for dependency relationships, we discover 40 distinct ways for the adversary to avoid detection. By submitting more constrained queries that make stronger assumptions, we can develop an understanding of what hoops an adversary is forced to jump through in those 40 possibilities. For example, if we additionally assume that neither of the components protected by Hyper-V are corrupted, there are only 24 possibilities. If, instead, we assumed only that no corruptions occur during the attestation (but allow components in Hyper-V to be corrupted) there are 12 possibilities. If we assume both that components in Hyper-V are not corrupted and that there are no other corruptions during the attestation, then the adversary cannot succeed.

While we believe constraining the adversary to performing recent or deep corruptions is often an acceptable risk, there may be circumstances where this assumption is unrealistic. Perhaps the signature file for the antivirus is not integrity protected. Modification of this file would affect the outcome of the virus scan. We would want to enrich the Copland phrase above to include a measurement of us.sigfile. However, it could be very easy for malware to remove its own signature from this unprotected list during the attestation. In our analysis, therefore, we would preclude all corruptions during the attestation except for corruptions of the signature. We would write:

$$\begin{aligned}\forall e_1, e_2, p, c\ .\ e_1 \prec e_2 \land \ell(e_2) = \mathsf{cor}(p.c) \land \mathsf{ms\_evt}(e_1)\\ \Rightarrow p = \mathsf{us} \land c = \mathsf{sigfile}\end{aligned} \qquad (8)$$

In this way, Chase run with our layered attestation theory provides an interactive method for exploring the trust consequences of Copland phrases. There are often many ways to collect any given set of measurements. By running Chase on the Copland phrases representing the variety of measurement strategies, we can understand the relative strengths and weaknesses among them. The fewer assumptions that need to be made in order to guarantee successful detection of any corruptions, the stronger the Copland phrase. Since there will never be a single solution to fit all use cases, we believe this exploratory approach to analyzing the trustworthiness of layered attestation strategies is an essential capability in designing attestation systems & protocols and in selecting sets of Copland phrases suitable for given situations.

7.3 Performance and Scaling

A natural concern with any automated analysis is whether it can terminate quickly, and how the performance scales with size. In our case, the analysis is meant to be applied at design time. Ideally, an analyst would use this approach to "debug" a Copland phrase as they decide what to include in an attestation policy. Since this analysis is not performed at runtime, we easily tolerate a search that takes seconds or even a few minutes to complete.

Another important aspect of interpreting performance results is that an incomplete search that finds models to inspect is also of value to the analyst. It already highlights areas of weakness that can be addressed either by modifying the phrase itself, or by ensuring the target system has mechanisms to support the types of assumptions that help constrain the search. The latter point is key, because a single search constraint justified by a single defensive mechanism will typically eliminate a significant fraction of the models from the search. Thus, even when a search finds many models, it is often sufficient to inspect only a few of them to determine useful search constraints resulting in a more reasonable set of models.

There are several factors that affect the performance of Chase on our examples: details of the phrase being analyzed, constraints used during the search, Chase's scheduling of formulas, and the inclusion of non-minimal models. Before reporting the detailed performance numbers for the examples presented so far, we discuss these factors as well as some optimizations we have developed.

First, most issues with performance stem from the way in which formulas with disjunction create new models in the search. Formulas with disjunction create branches in the search tree, so phrases that exercise these formulas more tend to generate larger search trees and take longer to analyze. Due to the branching implicit in Def. 5 (adversary-ordered) phrases that use a lot of parallelism (the ~ operator) tend to take longer than those with minimal branching. The formulas axiomatizing the definition of adversary-ordered offer several ways to resolve the orders between events that are unconstrained by the parallelism. Thus, performance does not scale simply with the size of a phrase alone.

Related to the above is the fact that searches with no constraints on the Depends relation result in more models and take longer to complete than those with very explicit dependencies. One of the branches of Formula 1 in Fig. 6 contains an existential variable representing a component. This variable may represent a component not yet known, or it may represent one of the components involved in other events. This indeterminacy often leads to further branching in the search. Explicit dependency relations help manage this indeterminacy by ensuring the existentially quantified variable gets equated with a fixed constant. Of course the other constraints presented above that rule out recent or deep corruptions will dramatically reduce the number of models of the theory, and hence result in faster analyses as well.

It turns out that the way in which Chase schedules formulas to apply can have a dramatic outcome on the size of the search tree. By default, Chase relies on fair scheduling to ensure that every formula has the opportunity to be applied. Our runs of Chase overrule its default scheduling to help us more quickly detect branches that will not lead to models. While we had to rely on detailed domain knowledge to determine a schedule that performs well in practice, the scheduling seems to be most sensitive to the order of rules in the fixed (non-phrase-specific) theory, so others using this approach can already benefit from a schedule that works well in practice.

Finally, although Chase is guaranteed to find a set of support, it does not guarantee that all the models it finds will be minimal. In our early development, we found that Chase was discovering models with "extraneous" events. These fell into two clear categories. First is when $\mathsf{cor}(p.c)$ is followed immediately by $\mathsf{rep}(p.c)$ with no measurement event between them to "witness" the corruption. Such a model is not minimal because the model obtained by removing those two events still satisfies the theory and is smaller (in the sense described in Section 5). Second, models with two instances of $\mathsf{cor}(p.c)$ without a $\mathsf{rep}(p.c)$ event in between are not minimal because the second corruption event is redundant. These two examples lead to four instances of non-minimality, two for each example because the roles of $\mathsf{cor}(p.c)$ and $\mathsf{rep}(p.c)$ can be reversed. We therefore added the four formulas depicted in Fig. 9 to our theory to preclude models with such instances of non-minimality.

(∗ Addresses pointless alternating corruptions and repairs ∗)

$$\forall e_1, e_2, p, c \,.\, \ell(e_1) = \mathsf{cor}(p.c) \land \ell(e_2) = \mathsf{rep}(p.c) \land e_1 \prec e_2$$
$$\Rightarrow \exists e_3 \,.\, e_1 \prec e_3 \land e_3 \prec e_2 \land \mathsf{ms\_evt}(e_3) \land \mathsf{relevant}(p.c, e_3)$$

$$\forall e_1, e_2, p, c \,.\, \ell(e_1) = \mathsf{rep}(p.c) \land \ell(e_2) = \mathsf{cor}(p.c) \land e_1 \prec e_2$$
$$\Rightarrow \exists e_3 \,.\, e_1 \prec e_3 \land e_3 \prec e_2 \land \mathsf{ms\_evt}(e_3) \land \mathsf{relevant}(p.c, e_3)$$

(∗ Addresses redundant corruptions and repairs ∗)

$$\forall e_1, e_2, p, c \,.\, \ell(e_1) = \mathsf{cor}(p.c) \land \ell(e_2) = \mathsf{cor}\, p.c \land e_1 \prec e_2$$
$$\exists e_3 \,.\, e_1 \prec e_3 \land e_3 \prec e_2 \land \ell(e_3) = \mathsf{rep}(p.c)$$

$$\forall e_1, e_2, p, c \,.\, \ell(e_1) = \mathsf{rep}(p.c) \land \ell(e_2) = \mathsf{rep}\, p.c \land e_1 \prec e_2$$
$$\exists e_3 \,.\, e_1 \prec e_3 \land e_3 \prec e_2 \land \ell(e_3) = \mathsf{cor}(p.c)$$

Figure 9: Formulas to eliminate non-minimal models.

Combining the careful scheduling of Chase with the formulas of Fig. 9, we obtain the performance results shown in Table 2. The results reported are based on runs using a 2018 MacBook Air with 1.6GHz Dual-Core Intel Core i5 processor with 16GB of RAM. These times represent real time experienced by the user accounting for all surrounding computations including compiling the Copland phrase into Chase input and processing the results to be visualized in a web browser.

Table 2: Performance results. The columns indicate (in order): the phrase analyzed, whether explicit dependency relation is given, whether deep corruptions are excluded, whether recent corruptions are excluded, the number of models found, total elapsed time.

Phrase	Expl. Dep.	Exclude Deep	Exclude Recent	# Models Found	Time
Ex. 1				5	0.53s
Ex. 1	✓			3	0.42s
Ex. 1	✓	✓	✓	1	0.33s
Ex. 2				4	0.48s
Ex. 2	✓			2	0.41s
Ex. 2	✓	✓	✓	0	0.31s
Ex. 17				2,478	4m15.08s
Ex. 17	✓			40	2.64s
Ex. 17	✓	✓	✓	0	0.51s

In the worst case, (Example 17 run without any constraints) the search terminates and discovers 2,478 models in 4 minutes 15 seconds. If the search is artificially limited to inspecting only 5,000 structures, Chase can inspect all these in under 10 seconds. However, as reported above, as soon as a dependency relation is explicitly represented, the theory only admits 40 models which are found in under 3 seconds. If recent and deep corruptions are excluded, Chase exhausts it search in about 0.5 seconds without finding any models.

8 RELATED WORK

Too much has been written about remote attestation to perform a comprehensive literature review, so we contrast our work with the most relevant examples of related efforts in the literature.

One central tenet of the approach we take is that any fixed set of attestation solutions will be insufficient to accommodate the inevitable variety encountered in the type and architecture of target devices and the contexts in which the trust decisions they support will be made. This observation is not new. This was implicit in [3] which develops a set of principles that should guide the design and implementation of remote attestation systems. More recently, the IETF has formed a working group focused on Remote ATtestation procedureS (RATS). Their draft architecture document [2] explicitly acknowledges the need for flexibility of mechanisms in implementing remote attestations. They also similarly envision the use of layered architectures to further enhance the quality of attestations. Similarly, Copland was designed to enable flexible specifications of complex attestations [19]. A complementary body of work seeks to develop flexible implementations that can be adapted to a wide range of scenarios [14, 16, 17]. The flexibility allowed for by both design and implementation leaves room for undesirable configurations that can yield untrustworthy results. We believe that our analysis methodology complements such work by offering a means for assessing the risk of any given option by understanding what an adversary must do to defeat it.

Of course, our work assumes that the target system has certain measurement capabilities and that they are reasonably effective at detecting corruption of subcomponents. Plenty has been written about particular approaches to measuring individual components ranging from flexible and programmable solutions [8, 24] to fixed solutions designed for particular architectures or components [5, 11, 12]. That line of research lies below the level of abstraction used in our execution model. However, it would be interesting to investigate whether the ideas we develop here could be suitably adapted to analyzing the measurement strategies for particular components. In particular, it may be possible to combine Copland [19] with the MSRR specification language for measurements [8] to obtain a top-to-bottom specification. By adapting the methods presented here, one would then have an accompanying top-to-bottom trust analysis as well.

Although many approaches take into account the possibility of an adversary interfering, few explicitly consider a runtime adversary that can interfere *during* an attestation. There are a few exceptions, most notably [21] which is the basis of the current work. The adversary model was enriched in [20] to account for manipulation of evidence in transit, a possibility not explored by the current work. Another analytical framework for analyzing attestations can be found in [4] in which they account for the effects of corruption on an attestation. The primary difference is that runtime corruptions are not considered. Additionally, the analysis is focused on Intel's "late launch" capability. None of these analytic frameworks have any automated support. To the best of our knowledge the current work is the first automated analysis methodology for remote attestation.

9 CONCLUSION

In this paper we introduced an approach to automated trust analysis of layered attestation protocols. We introduce a tool chain that ingests the specification of a layered attestation protocol written in the Copland language together with a configurable set of system assumptions, and produces a characterization of all the ways an active adversary might avoid detection by the attestation protocol. Our methodology presents opportunities for designers and implementers alike in developing attestation protocols and understanding how they help buy down risk when used in support of trust decisions.

Our approach is to compile a Copland specification into a partially ordered set of measurement events and apply our general-purpose model finder for geometric logic, called Chase, to characterize all the ways an active adversary can avoid detection under a given set of assumptions. This strategy requires us to equip Chase with a first-order logical theory that axiomatizes our execution model in the presence of an active adversary. We develop the theory of *saturated queries* and demonstrate that this theory correctly captures our intended denotation (Theorem 12). We present an axiomatization of the theory of saturated queries and prove that this axiomatization is correct (Theorem 16).

While the adversary model used in this paper is a useful one, it does not capture all the ways to interfere with the expected functioning of a layered attestation. We believe a fruitful line of future research would be to expand the adversary model to account for the ability to tamper with the integrity of evidence and to alter the control flow of an attestation. Such capabilities are reminiscent of the abilities of a network adversary assumed in the analysis of cryptographic protocols. A promising direction would be to determine how to integrate analyses using Chase with analyses using cryptographic protocol analysis tools. This would allow for a more complete understanding of the true benefits provided by well-designed layered attestations.

REFERENCES

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[2] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan. Remote attestation procedures architecture, 2021. https://datatracker.ietf.org/doc/draft-ietf-rats-architecture/ (Accessed 22-Feb-2021).

[3] George Coker, Joshua D. Guttman, Peter Loscocco, Amy L. Herzog, Jonathan K. Millen, Brian O'Hanlon, John D. Ramsdell, Ariel Segall, Justin Sheehy, and Brian T. Sniffen. Principles of remote attestation. *Int. J. Inf. Sec.*, 10(2):63–81, 2011.

[4] Anupam Datta, Jason Franklin, Deepak Garg, and Dilsun Kirli Kaynar. A logic of secure systems and its application to trusted computing. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 221–236, 2009.

[5] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In *Proceedings of the 4th ACM Workshop on Scalable Trusted Computing, STC 2009, Chicago, Illinois, USA, November 13, 2009*, pages 49–54, 2009.

[6] Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, 2001.

[7] John Fisher and Marc Bezem. *Geolog and Skolem Machines.* California State Polytechnic and University of Bergen, 2007. https://www.cpp.edu/~jrfisher/www/prolog_tutorial/logic_topics/geolog/index.html.

[8] Jason Gevargizian and Prasad Kulkarni. MSRR: measurement framework for remote attestation. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2018, Athens, Greece, August 12-15, 2018*, pages 748–753. IEEE Computer Society, 2018.

[9] Trusted Computing Group. TPM Main Specification Level 2 version 1.2, Parts 1–3, Revision 116, 2011. https://trustedcomputinggroup.org/resource/tpm-main-specification/.

[10] Intel. Intel® Software Guard Extensions (Intel® SGX), 2016. https://software.intel.com/en-us/sgx.

[11] Chongkyung Kil, Emre Can Sezer, Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009, Estoril, Lisbon, Portugal, June 29 - July 2, 2009*, pages 115–124, 2009.

[12] Peter Loscocco, Perry W. Wilson, J. Aaron Pendergrass, and C. Durward McDonell. Linux kernel integrity measurement using contextual inspection. In *Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, STC 2007, Alexandria, VA, USA, November 2, 2007*, pages 21–29, 2007.

[13] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[14] J. Aaron Pendergrass, Sarah Helble, John Clemens, and Peter Loscocco. A platform service for remote integrity measurement and attestation. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, 2018.

[15] Adam Petz. Personal communication, 2021.

[16] Adam Petz and Perry Alexander. A copland attestation manager. In Xenofon D. Koutsoukos, Alvaro A. Cárdenas, and Ehab Al-Shaer, editors, *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security, HotSoS 2019, Nashville, TN, USA, April 1-3, 2019*, pages 6:1–6:10. ACM, 2019.

[17] Adam Petz and Perry Alexander. An infrastructure for faithful execution of remote attestation protocols. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods*, pages 268–286, Cham, 2021. Springer International Publishing.

[18] John D. Ramsdell. *Chase Source Repository.* The MITRE Corporation, 2019. https://github.com/ramsdell/chase, install with `opam install chase`.

[19] John D. Ramsdell, Paul D. Rowe, Perry Alexander, Sarah C. Helble, Peter Loscocco, J. Aaron Pendergrass, and Adam Petz. Orchestrating layered attestations. In Flemming Nielson and David Sands, editors, *Principles of Security and Trust*, volume 11426, pages 197–221, Cham, 2019. Springer International Publishing.

https://ku-sldg.github.io/copland/resources/copland-post-2019.pdf.

[20] Paul D. Rowe. Bundling evidence for layered attestation. In Michael Franz and Panos Papadimitratos, editors, *Trust and Trustworthy Computing*, pages 119–139, Cham, 2016. Springer International Publishing.

[21] Paul D. Rowe. Confining adversary actions via measurement. In Barbara Kordy, Mathias Ekstedt, and Dong Seong Kim, editors, *Graphical Models for Security*, pages 150–166, Cham, 2016. Springer International Publishing.

[22] Paul D. Rowe, John D. Ramsdell, and Ian D. Kretz. Copland: Semantics, languages and tools for layered attestation, 2021. https://copland-lang.org/resources/chase/ppdp/README.

[23] Salman Saghafi and Daniel J. Dougherty. Razor: Provenance and exploration in model-finding. In *4th Workshop on Practical Aspects of Automated Reasoning (PAAR)*, 2014.

[24] Mark Thober, J. Aaron Pendergrass, and Andrew D. Jurik. JMF: Java measurement framework. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, pages 21–32. ACM, 2012.

[25] Anthony Velte and Toby Velte. *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc., 2009.