Policy Preferences for Copland Phrases

Whoever wants to contribute

No Institute Given

Abstract. This is a living document aimed at describing an approach to rate the relative strength of alternative Copland phrases [2]. We motivate the construction of a ranking function ρ to order phrases. We then outline a general approach to constructing useful ranking functions based on characterizing the set of possible executions allowed by a Copland phrase in the presence of an adversary.

1 Statement of Purpose

The high-level goal of this work is to develop a framework for ranking Copland phrases. The broader impact will be to help users of Maat and related frameworks write local policies to choose among available Copland phrases (i.e. APBs). We imagine Maat users will do any relevant analyses at registration time to alter their policies governing which Copland phrases are preferred under various contexts and scenarios.

Creating a ranking of phrases amounts to imposing an order relation $\preceq$ on them. Equivalently, we think of it as finding an ordered set $(P, \leq)$ and a ranking function $\rho : \textsc{Copland} \to P$. This naturally imposes an ordering on phrases by the following rule:

$$T_1 \preceq T_2 \quad \text{iff } \rho(T_1) \leq \rho(T_2) \tag{1}$$

Pitfalls to avoid. The process of finding such a ρ would be trivial if we were willing to accept absolutely any ordering $\preceq$. For example, we could use some kind of lexicographic ordering on phrases. This obviously provides no value to Maat users because it does not relate to the Copland semantics in any way. At the other end of the spectrum, it would also be a mistake to fix $\preceq$ too rigidly, once and for all. We must be able to account for differences in context and differences of opinion or preference. One appraiser may wish to focus on raising the bar as high as possible for an adversary, while another may want to ensure the phrase executes within some given timeframe, so they will not accept "strong" phrases that take too long. We are therefore interested in developing a *framework* for constructing ranking functions that is semantically grounded, but leaves room for individual preferences.

Unifying principle. What does it mean for the construction of ρ to be semantically grounded? We take the stance that, regardless of personal preferences, sound reasons for preferring one phrase to another will be based on an understanding of

1. The intended execution semantics of the phrase (as developed in [2]).
2. The ways in which an adversary might be able to alter the intended executions.

These two points motivate an approach in which we construct ρ first by analyzing the phrase to determine the set of executions it allows (accounting for adversary behavior), and then applying a ranking function on the resulting set based on the features of the possible executions. This is schematized as follows:

$$\textsc{Copland} \longrightarrow \text{Executions} \longrightarrow (P, \leq) \tag{2}$$

The analysis implied by the first arrow of (2) can be further decomposed into two steps. The first is a simple application of Copland's denotational (event poset) semantics. The second is a major focus of the rest of this work. It represents a complete characterization of possible ways an adversary can affect the outcome of various events in the semantics. Our approach will be to use a general purpose model finder to produce the desired characterization of possible executions. This overall approach schematized as follows:

$$\textsc{Copland} \longrightarrow \begin{array}{c}\text{Event Poset}\\(\text{no adversary})\end{array} \longrightarrow \begin{array}{c}\text{Executions}\\(\text{w/ adversary})\end{array} \longrightarrow (P, \leq) \tag{3}$$

The second arrow of (3) was the content of [4] as it applies to event posets consisting of only measurement events. That paper also included an instantiation of the third arrow in which P was chosen to be the two-point lattice $\{\top, \bot\}$. In particular, an execution was mapped to $\top$ iff every undetected corruption was either "recent" or "deep" for specific formal definitions of those two terms. In general, using the two-point lattice amounts setting a threshold of some kind and treating equally all the phrases that surpass the threshold. The "recent or deep" criterion is an interesting and natural threshold, but it is certainly not the only one possible.

The ideas presented in [3] are slightly different, in that they are meant to help an appraiser infer something about the structure of the event poset from the structure of the bundled evidence. They don't naturally correspond to any of the arrows in the above diagram. We have been exploring other ideas about the role of bundling events that fit more naturally into the flow described above. This will be a big focus below.

A note on lattices. Before proceeding to an overview of the model finding approach, we make an observation about the third arrow of (3). This represents two kinds of maps: one map, π, that takes individual executions and maps them to points in P, and another one, Π, that takes sets of executions (i.e. characterizations of all possible executions of a given term) and maps the set to a point in P. If we are given the map π, there is a natural way to define Π by assuming the adversary can always choose the worst option (i.e. lowest in the ordering). More formally, if E is a set of exectuions, we could define

$$\Pi(E) := glb(\{\pi(e) \mid e \in E\}) \tag{4}$$

where *glb* is the greatest lower bound. In general, the *glb* is not necessarily defined. However, if P is a (lower) semilattice then greatest lower bounds always exist by definition. This rule gives us strong motivations to choose lower semilattices for P.

2 Model Finding with the Chase

Chase 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.

$$\forall \boldsymbol{x}.\, P_1(\boldsymbol{x}) \wedge \cdots \wedge P_n(\boldsymbol{x}) \Rightarrow \bigvee_i \exists \boldsymbol{y}_i.\, Q_{i,1}(\boldsymbol{x}, \boldsymbol{y}_i) \wedge \cdots \wedge Q_{i,n_i}(\boldsymbol{x}, \boldsymbol{y}_i)$$

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 [1] 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 1 (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,*

$$P(a_1, \ldots, a_n) \in A \text{ implies } P(h(a_1), \ldots, h(a_n)) \in B.$$

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 2 (Minimal Model) *Model A of T is* minimal *iff forall models B of T, whenever $B \ll A$ then $A \ll B$.*

Definition 3 (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.

2.1 Input Syntax

The input to the Chase program is a set of sequents written in a slight variant of Geolog[1] syntax. The syntax is inspired by Prolog. Quantification is implicit and constants and variables are distinguished using capitalization, where variables are capitalized.

A Chase sequent has the form

$$A_1 \texttt{ \& } A_2 \texttt{ \& } \cdots \texttt{ \& } A_m \texttt{ => } C_1 \texttt{ | } C_2 \texttt{ | } \cdots \texttt{ | } C_n \texttt{.}$$

The formula to the left of `=>` is the antecedent, and the consequent is to the right. Each conjunct A_i in the antecedent is an atomic formula. The consequent is a disjunction. Each disjunct C_j is a conjunction of the form

$$B_{j,1} \texttt{ \& } B_{j,2} \texttt{ \& } \cdots \texttt{ \& } B_{j,p_j}$$

where each conjunct $B_{j,k}$ is an atomic formula.

A symbol is a letter followed by a sequence of letters, dollar signs ($), and underscores (_). An atomic formula is a predicate symbol (a symbol not capitalized) applied to a parenthesized sequence of comma separated terms, or an equality consisting of two terms separated by the = sign. A term is a variable (a capitalized symbol), a constant (a symbol not capitalized), or a function symbol (a symbol not capitalized) applied to a parenthesized sequence of comma separated terms.

An empty antecedent can be optionally indicated with the reserved symbol `true`. An empty consequent can be optionally indicated with the reserved symbol `false`. Comments start with `%` and continue until the end of the line. An example of a theory for input follows.

```
% Total Ordering
true => num(a) & num(b).
num(X) & num(Y) => lt(X, Y) | X = Y | lt(Y, X).
```

The variables that occur in the antecedent of a sequent are universally quantified. The variables that occur in disjunct B_j and not in the antecedent are existentially quantified over the disjunct. Thus,

```
true => p(X) | q(X).
```

means $(\exists X.\, p(X)) \vee (\exists X.\, q(X))$ *not* $\exists X.\, (p(X) \vee q(X))$.

To specify that X is universally quantified in the consequent, add `X=X` to the antecedent. Thus $\forall X.\, p(X)$ is written as

```
X = X => p(X).
```

Within this paper, we typeset theories using connectives from mathematical logic. Thus the Total Ordering Example from above becomes

[1] `https://www.cpp.edu/~jrfisher/www/prolog_tutorial/logic_topics/geolog/index.html`

```
% Total Ordering
true ⇒ num(a) ∧ num(b).
num(X) ∧ num(Y) ⇒ lt(X, Y) ∨ X = Y ∨ lt(Y, X).
```

2.2 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. $P(c_1, \ldots, c_n)$, or
2. $f(c_1, \ldots, c_n) = c_0$.

where P and f are in the signature of T.

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

1. $P(c_1, \ldots, c_n) \in A$ implies $c_1, \ldots, c_n \in U$, and
2. $f(c_1, \ldots, c_n) = c_0 \in A$ implies $c_0, \ldots, c_n \in U$.

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. Equality in A is closed under congruence.
3. Each element in U is the canonical representative of an equivalence class induced by congruence closure.
4. Each constant in C is the left-hand side of one equation, and its right-hand side is a constant in C.

One of the three models found by Chase for the Total Ordering Example is

$$\{\mathtt{num(a)}, \mathtt{num(b)}, \mathtt{lt(a,b)}, \mathtt{a} = \mathtt{a}, \mathtt{b} = \mathtt{b}\}.$$

2.3 Algorithm

Models of theory T are found using an algorithm called the chase. 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.

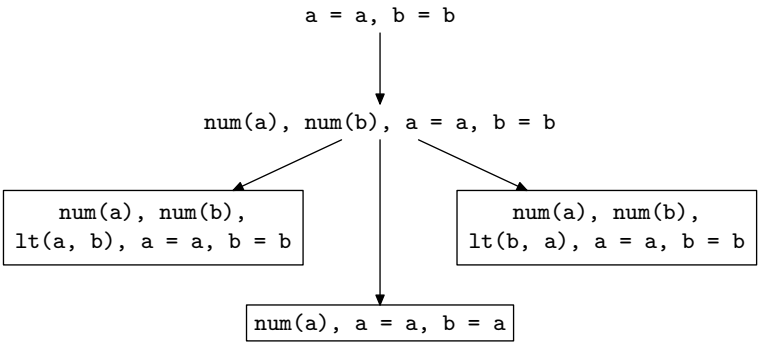

Fig. 1. Structures Generated From the Total Ordering Example

The structures generated by a run of Chase on the Total Ordering Example are shown in Fig. 1. 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 the 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.

3 Chase Theory for Confining Adversaries

This section will summarize what we have done so far regarding the implementation of the principles of layered attestation as laid out in [4]. It will give highlights of the chase theory (and possibly include the full theory as an appendix, or at least point our research team to the right place in the repository). We will also demonstrate what the theory produces in several variants of our typical example: when doing things top-down instead of bottom-up, and also in adding a configuration file for the virus checker that should also be measured. Perhaps this section will include some discussion points surrounding the third arrow in equation (3). In particular, are there other reasonable choices for P that are not all about recent or deep corruptions? Could we make sense of an estimate of total work required by the adversary if an appraiser wished to use such a thing?

4 Formal Details

Definition 1 (Measurement Systems). *We define a* measurement system *to be a tuple $\mathcal{MS} = (O, M, C)$, where O is a set of objects (e.g. software components) with a distinguished element* rtm*. M and C are binary relations on O. We call*

M *the* measures *relation, and*
C *the* context *relation.*

We say M *is* rooted *when for every* $o \in O \setminus \{\mathsf{rtm}\}$, $M^+(\mathsf{rtm}, o)$, *where* M^+ *is the transitive closure of* M. *We assert* $\bar{C}(o_1, o_2)$ *iff* $o_1 = o_2$ *or* $C(o_1, o_2)$.

Definition 2 (Labeled Poset). *A* labeled poset $(E, \prec, \ell)$ *is a set of events, a irreflexive, transitive relation* $\prec$ *on the events, and a function* $\ell : E \to L$ *that associates a label with each event.*

Definition 3 (Labels). *The set of labels* L *is the smallest set such that*

1. $\mathsf{att\text{-}start} \in L$,
2. $\mathsf{ms}_{\{o \mid C(o,o_1)\}}(o_1, o_2) \in L$ *if* $o_1, o_2 \in O$ *and* $M(o_1, o_2)$, *and*
3. $\mathsf{cor}(o) \in L$ *and* $\mathsf{rep}(o) \in L$ *if* $o \in O \setminus \{\mathsf{rtm}\}$.

1. *The* attestation start *event is labeled by* $\mathsf{att\text{-}start}$.
2. *A* measurement event *is labeled by* $\mathsf{ms}_{\{o \mid C(o,o_1)\}}(o_1, o_2)$ *such that* $M(o_1, o_2)$. *We say such an event* measures o_2, *and we call* o_2 *the* target *of* e. *When* $C^{-1}(o_1)$ *is empty we omit the subscript and write* $\mathsf{ms}(o_1, o_2)$.
3. *An* adversary event *is labeled by either* $\mathsf{cor}(o)$ *or* $\mathsf{rep}(o)$ *for* $o \in O \setminus \{\mathsf{rtm}\}$.

Definition 4 (Event Classifications). *Let* $P = (E, \prec, \ell)$. *The set of measurement events in* P *is* $\mathcal{ME}(P) = \{e \mid \ell(e) = \mathsf{ms}_{\{o \mid C(o,o_1)\}}(o_1, o_2)\}$. *The set of adversary events in* P *is* $\mathcal{AE}(P) = \{e \mid \ell(e) = \mathsf{cor}(o) \vee \ell(e) = \mathsf{rep}(o)\}$.

Definition 5 (Touches). *In poset* $P = (E, \prec, \ell)$, *event* $e \in E$ touches *object* $o \in O$, *written* $\mathcal{T}_P(e, o)$, *iff*

1. $\ell(e) = \mathsf{cor}(o)$,
2. $\ell(e) = \mathsf{rep}(o)$,
3. $\ell(e) = \mathsf{ms}_{\{o \mid C(o,o_1)\}}(o_1, o_2)$ *and* $\bar{C}(o, o_1)$ *or* $o = o_2$.

Let $\mathcal{T}_P(o) = \{e \mid \mathcal{T}_P(e, o)\}$ *be the set of events that touch* o *in* P.

Definition 6 (Adversary-Ordered). *Poset* $P = (E, \prec, \ell)$ *is* adversary-ordered *iff for all* $o \in O$, *and* $e_1, e_2 \in \mathcal{T}_P(o)$, *if* e_1 *is an adversary event and* $e_1 \neq e_2$, *then either* $e_1 \prec e_2$ *or* $e_2 \prec e_1$.

Definition 7 (Execution). *An* execution $X = (E, \prec, \ell)$ *is a finite, adversary-ordered, labeled poset.*

Lemma 1. *Consider some object* $o \in O$. *The adversary events that touch* o *in execution* X *is* $A_o = \mathcal{T}_X(o) \cap \mathcal{AE}(X)$. *For some non-adversarial event* $e \in \mathcal{T}_X(o)$, *let* $e{\downarrow}$ *be the set of events in* A_o *that precede* e. *The set* $e{\downarrow}$ *is empty or has exactly one maximal element.*

Proof. Suppose $e{\downarrow}$ is not empty. Since the set of events is finite, it has at least one maximal element. Suppose e' and e'' are distinct maximal elements. However, since X is adversary-ordered and e' is an adversary event, $e' \prec e''$ or $e'' \prec e'$, yielding a contradiction. □

Definition 8 (Corruption State). *Let $X = (E, \prec, \ell)$ be an execution. For each object $o \in O$, its corruption state is described by the function $cs(o) : \mathcal{T}_X(o) \to \{\mathsf{c}, \mathsf{r}\}$ where*

$$cs(o)(e) = \begin{cases} \mathsf{c} & : \ell(e) = \mathsf{cor}(o) \\ \mathsf{r} & : \ell(e) = \mathsf{rep}(o) \\ \mathsf{r} & : e \in \mathcal{ME}(X) \wedge e{\downarrow} = \emptyset \\ cs(o)(e') & : e \in \mathcal{ME}(X) \wedge e' \text{ maximal in } e{\downarrow} \end{cases}$$

where $A_o = \mathcal{T}_X(o) \cap \mathcal{AE}(X)$, and $e{\downarrow}$ is the set of events in A_o that precede e. We write $cs^X(o)(e)$ when we want to be explicit about the execution used to define the function. The type of function cs is the dependent type $\prod_{o \in O} \mathcal{T}_X(o) \to \{\mathsf{c}, \mathsf{r}\}$.

Definition 9 (Partial Measurement Output). *For each object o, let $\mathcal{G}(o)$ and $\mathcal{B}(o)$ be a partition of $\mathcal{MV}(o)$, the set of potential measurement values for o. Let $X = (E, \prec, \ell)$ be an execution and let*

$$\mathcal{M}_X(o_2) = \{e \in E \mid \ell(e) = \mathsf{ms}_{\{o \mid C(o,o_1)\}}(o_1, o_2)\},$$

the set of events that measure o_2 in X. For each object o, its output is described by the function $out(o) : \mathcal{M}_X(o) \to \mathcal{MV}(o)$, where $out(o)(e) = v$ iff there exists object o_1 such that $\ell(e) = \mathsf{ms}_{\{o \mid C(o,o_1)\}}(o_1, o_2)$ and either

1. *$v \in \mathcal{B}(o_2)$, $cs(o_2)(e) = \mathsf{c}$, and $\forall o \,.\, \bar{C}(o, o_1)$ implies $cs(o)(e) = \mathsf{r}$, or*
2. *$v \in \mathcal{G}(o_2)$.*

We write $out^X(o)(e)$ when we want to be explicit about the execution used to define the function. The type of function out is the dependent type $\prod_{o \in O} \mathcal{M}_X(o) \to \mathcal{MV}(o)$.

If $out(o)(e) \in \mathcal{B}(o)$ we say *e detects a corruption* of o. If $out(o)(e) \in \mathcal{G}(o)$ but $cs(o)(e) = \mathsf{c}$, we say the adversary *avoids detection at e.*

The bundle-like objects are (X, cs^X, out^X).

JDR: Did I fix homomorphisms now?

Here we fix the partitions $\mathcal{B}$ and $\mathcal{G}$.

Definition 10 (Homomorphism). *For executions $X_1 = (E_1, \prec_1, \ell_1)$ and $X_2 = (E_2, \prec_2, \ell_2)$, the triple*

$$\eta : (X_1, cs_1, out_1) \mapsto (X_2, cs_2, out_2)$$

is a homomorphism iff

1. *$\eta : E_1 \to E_2$,*
2. *η is injective,*
3. *η preserves labels: $\ell_1(e) = \ell_2(\eta(e))$,*
4. *η preserves orderings: $e_1 \prec_1 e_2$ implies $\eta(e_1) \prec_2 \eta(e_2)$,*
5. *$e \in \mathcal{T}_{X_1}(o)$ implies $cs_1(o)(e) = cs_2(o)(\eta(e))$, and*
6. *$e \in \mathcal{M}_{X_1}(o)$ implies $out_1(o)(e) = out_2(o)(\eta(e))$.*

PDR: Here again, let's be more explicit about the function types for cs_i and out_i. For a labeled poset $(E, \prec, \ell)$, define $T_E := \{(e, o) \mid e \text{ touches } o\}$. Then $cs_i : T_{E_i} \to \{\mathsf{c}, \mathsf{r}\}$ is a partial function. Similarly, $out_i : M_E \to \{\mathsf{X}, \checkmark\}$ is a partial function. Then for both conditions 5 and 6, the required equality should be guarded by the input being in the domain of the function.

Notice, that out_i in the Def. 14 is not necessarily the same as out_i defined in Def. 13. The ones in Def. 14 are arbitrary partial assignments. The one is Def. 13 is a total function that is derived from the labels and the corruption states (values of cs_E, which is also a total function in that case). The collision of function identifiers is confusing. We should call them something else when they are the partial functions appearing in specifications.

* * *

5 Chase Theory for Bundling

The underlying ideas for this section are much less solidified. This section needs to do several things. First, we should introduce the basic motivation for the theory that is all about chain of custody. That is, we identify a few key things that can go wrong if evidence is not integrity protected when it is handled by a corrupt component. The motivation leads to the need to represent evidence *values*, not just evidence types. So we can copy over here the modified evidence semantics from the `ev/` directory. This section should also include the 4 Copland phrases we used as intuition builders for the core ideas here. Eliciting and encoding the group's intuitions about how these 4 phrases *should* be ordered will be crucial for deciding what should be in the chase theory.

6 Formal Details

$$
\begin{aligned}
\mathsf{M} &\leftarrow \mathsf{r} \mid \mathsf{c} \mid v_{\mathsf{Int}} \\
\mathsf{EV} &\leftarrow \mathsf{mt} \mid \mathsf{kim}^{\mathsf{Int}}_{\mathsf{Int}}(\mathsf{EV}, \mathsf{M}) \mid \mathsf{usm}_{\mathsf{Int}}(\mathsf{EV}, \mathsf{M}) \\
&\quad \mid [\![\mathsf{EV}]\!]_{\mathsf{Int}} \mid \#(\mathsf{Int}, \mathsf{EV}) \mid (\mathsf{EV} \;;; \mathsf{EV}) \mid (\mathsf{EV} \parallel \mathsf{EV})
\end{aligned}
$$

Fig. 2. Evidence Value Grammar

$$
\begin{array}{l}
\mathsf{fv}(i, \mathsf{mt}) = (i, \mathsf{mt}) \\
\mathsf{fv}(i, \mathsf{usm}_p(e, M)) = \\
\quad \mathsf{let}\ (j, a) = \mathsf{fv}(i, e)\ \mathsf{in} \\
\quad (j + 1, \mathsf{usm}_p(a, v_j)) \\
\mathsf{fv}(i, \mathsf{kim}_p^q(e, M)) = \\
\quad \mathsf{let}\ (j, a) = \mathsf{fv}(i, e)\ \mathsf{in} \\
\quad (j + 1, \mathsf{kim}_p^q(a, v_j)) \\
\mathsf{fv}(i, [\![e]\!]_k) = \\
\quad \mathsf{let}\ (j, a) = \mathsf{fv}(i, e)\ \mathsf{in} \\
\quad (j, [\![a]\!]_k) \\
\mathsf{fv}(i, \#(p, e)) = \\
\quad \mathsf{let}\ (j, a) = \mathsf{fv}(i, e)\ \mathsf{in} \\
\quad (j, \#(p, a)) \\
\mathsf{fv}(i, e_1 \mathbin{;;} e_2) = \\
\quad \mathsf{let}\ (j, a_1) = \mathsf{fv}(i, e_1)\ \mathsf{in} \\
\quad \mathsf{let}\ (k, a_2) = \mathsf{fv}(j, e_2)\ \mathsf{in} \\
\quad (k, a_1 \mathbin{;;} a_2) \\
\mathsf{fv}(i, e_1 \parallel e_2) = \\
\quad \mathsf{let}\ (j, a_1) = \mathsf{fv}(i, e_1)\ \mathsf{in} \\
\quad \mathsf{let}\ (k, a_2) = \mathsf{fv}(j, e_2)\ \mathsf{in} \\
\quad (k, a_1 \parallel a_2)
\end{array}
$$

Fig. 3. Evidence Fresh Variables

$$
\begin{array}{rll}
\mathcal{E}''(i, \mathsf{CPY}, p, e) = (i, e) & & \\
\mathcal{E}''(i, \mathsf{msp}(\mathsf{USM}, q, t), p, e) = (j + 1, \mathsf{usm}_p(a, v_j)) & (j, a) = \mathsf{fv}(i, e) & \\
\mathcal{E}''(i, \mathsf{msp}(\mathsf{KIM}, q, t), p, e) = (j + 1, \mathsf{kim}_p^q(a, v_j)) & (j, a) = \mathsf{fv}(i, e) & \\
\mathcal{E}''(i, \mathsf{SIG}, p, e) = (i, [\![e]\!]_{\mathsf{sk}(p)}) & & \\
\mathcal{E}''(i, \mathsf{HSH}, p, e) = (i, \#(p, e)) & & \\
\mathcal{E}''(i, @_q\, t, p, e) = \mathcal{E}''(i, t, q, e) & & \\
\mathcal{E}''(i, t_1 \to t_2, p, e_1) = \mathcal{E}''(j, t_2, p, e_2) & (j, e_2) = \mathcal{E}''(i, t_1, p, e_1) & \\
\mathcal{E}''(i, t_1 \overset{\pi}{\prec} t_2, p, e) = (k, e_1 \mathbin{;;} e_2) & \pi = (\pi_1, \pi_2) & \\
 & (j, e_1) = \mathcal{E}''(i, t_1, p, \pi_1(e)) & \\
 & (k, e_2) = \mathcal{E}''(j, t_2, p, \pi_2(e)) & \\
\mathcal{E}''(i, t_1 \overset{\pi}{\sim} t_2, p, e) = (k, e_1 \parallel e_2) & \pi = (\pi_1, \pi_2) & \\
 & (j, e_1) = \mathcal{E}''(i, t_1, p, \pi_1(e)) & \\
 & (k, e_2) = \mathcal{E}''(j, t_2, p, \pi_2(e)) &
\end{array}
$$

Fig. 4. Evidence Value Semantics

$$
\begin{array}{rll}
\mathcal{V}''(\mathsf{CPY}, p, e, i, x) = (i, \mathsf{CPY}(i, p, e)) & & \\
\mathcal{V}''(\mathsf{msp}(\mathsf{USM}, q, t), p, e, i, x) = (i+1, \mathsf{USM}(i, p, q, e, e')) & & (y, e') = \mathcal{E}''(x, \mathsf{msp}(\mathsf{USM}, q, t), p, e) \\
\mathcal{V}''(\mathsf{msp}(\mathsf{KIM}, q, t), p, e, i, x) = (i+1, \mathsf{KIM}(i, p, q, e, e')) & & (y, e') = \mathcal{E}''(x, \mathsf{msp}(\mathsf{KIM}, q, t), p, e) \\
\mathcal{V}''(\mathsf{SIG}, p, e, i, x) = (i+1, \mathsf{SIG}(i, p, e, [\![e]\!]_{\mathsf{sk}(p)})) & & \\
\mathcal{V}''(\mathsf{HSH}, p, e, i, x) = (i+1, \mathsf{HSH}(i, p, e, \#(p, e))) & & \\
\mathcal{V}''(@_q\, t, p, e, i, x) = (j+1, \mathsf{REQ}(i, p, q, e) \triangleright & & \\
\qquad \mathsf{O} \triangleright \mathsf{RPY}(j, p, q, e')) & & (y, e') = \mathcal{E}''(x, t, q, e) \\
& & (j, \mathsf{O}) = \mathcal{V}''(t, q, e, i+1, x) \\
\mathcal{V}''(t_1 \to t_2, p, e, i, x) = (k, \mathsf{O}_1 \triangleright \mathsf{O}_2) & & (y, e') = \mathcal{E}''(x, t_1, p, e) \\
& & (j, \mathsf{O}_1) = \mathcal{V}''(t_1, p, e, i, x) \\
& & (k, \mathsf{O}_2) = \mathcal{V}''(t_2, p, e', j, y) \\
\mathcal{V}''(t_1 \overset{\pi}{\prec} t_2, p, e, i, x) = (k+1, \mathsf{SPLIT}(i, e, \pi_1(e), \pi_2(e)) \triangleright & & \\
\qquad \mathsf{O}_1 \triangleright \mathsf{O}_2 \triangleright \mathsf{JOIN}(k, e_1, e_2, e_1 \mathbin{;;} e_2)) & & \pi = (\pi_1, \pi_2) \\
& & (y, e_1) = \mathcal{E}''(x, t_1, p, \pi_1(e)) \\
& & (z, e_2) = \mathcal{E}''(y, t_2, p, \pi_2(e)) \\
& & (j, \mathsf{O}_1) = \mathcal{V}''(t_1, p, \pi_1(e), i+1, x) \\
& & (k, \mathsf{O}_2) = \mathcal{V}''(t_2, p, \pi_2(e), j, y) \\
\mathcal{V}''(t_1 \overset{\pi}{\sim} t_2, p, e, i, x) = (k+1, \mathsf{SPLIT}(i, e, \pi_1(e), \pi_2(e)) \triangleright & & \\
\qquad (\mathsf{O}_1 \bowtie \mathsf{O}_2) \triangleright \mathsf{JOIN}(k, e_1, e_2, e_1 \parallel e_2)) & & \pi = (\pi_1, \pi_2) \\
& & (y, e_1) = \mathcal{E}''(x, t_1, p, \pi_1(e)) \\
& & (z, e_2) = \mathcal{E}''(y, t_2, p, \pi_2(e)) \\
& & (j, \mathsf{O}_1) = \mathcal{V}''(t_1, p, \pi_1(e), i+1, x) \\
& & (k, \mathsf{O}_2) = \mathcal{V}''(t_2, p, \pi_2(e), j, y)
\end{array}
$$

Fig. 5. Event Value Semantics

A Minimum Confining Example

```
% Attestation example
% ********

%%%%%%%%%%%%%%%%%%%%%%%%
% Strict partial orders %
%%%%%%%%%%%%%%%%%%%%%%%%

% Irreflexivity:
prec(E, E) ⇒ false.                                          % (0)
ctxt(C, C) ⇒ false.                                          % (1)

% Transitivity:
prec(E1, E2) ∧ prec(E2, E3) ⇒ prec(E1, E3).                  % (2)
ctxt(C1, C2) ∧ ctxt(C2, C3) ⇒ ctxt(C1, C3).                  % (3)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mutually exclusive event types %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

E = ms(I1, C1, C2) ∧ E = cor(I2, C3) ⇒ false.                % (4)
E = ms(I1, C1, C2) ∧ E = rep(I2, C3) ⇒ false.                % (5)
E = cor(I1, C1) ∧ E = rep(I2, C2) ⇒ false.                   % (6)

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Relevance of components %
%%%%%%%%%%%%%%%%%%%%%%%%%%

% Introduction of relevant:
E = ms(I, C1, C2) ⇒ relevant(C1, E) ∧ relevant(C2, E).       % (7)
E = ms(I, C1, C2) ∧ ctxt(C3, C1)
  ⇒ relevant(C1, E) ∧ relevant(C2, E) ∧ relevant(C3, E).     % (8)
E = cor(I, C) ⇒ relevant(C, E).                              % (9)
E = rep(I, C) ⇒ relevant(C, E).                              % (10)

% Adversary ordered:
relevant(C, E1) ∧ relevant(C, E2) ∧ E1 = cor(I, C)
  ⇒ prec(E1, E2) ∨ prec(E2, E1) ∨ E1 = E2.                   % (11)
relevant(C, E1) ∧ relevant(C, E2) ∧ E1 = rep(I, C)
  ⇒ prec(E1, E2) ∨ prec(E2, E1) ∨ E1 = E2.                   % (12)

%%%%%%%%%%%%%%%%%%%%%%%%
% Facts about lastAdvEv %
%%%%%%%%%%%%%%%%%%%%%%%%

lastAdvEv(E0, E1) ∧ prec(E0, cor(I, C)) ∧ prec(cor(I, C), E1)
  ⇒ false.                                                   % (13)
lastAdvEv(E0, E1) ∧ prec(E0, rep(I, C)) ∧ prec(rep(I, C), E1)
```

```
  ⇒ false.                                                          % (14)

%%%%%%%%%%%%%%%%%%%%%%%%
% Logic of corruption. %
%%%%%%%%%%%%%%%%%%%%%%%%

% Root of trust is never corrupt
% (Not essential to the theory).
cor_at(rtm, E) ⇒ false.                                             % (15)

% Being corrupt or not are mutually exclusive.
cor_at(C, E) ∧ rep_at(C, E) ⇒ false.                                % (16)

% Assumption 1 (Measurement Accuracy):
ms_ok(ms(I, C1, C2)) ∧ cor_at(C2, ms(I, C1, C2))
  ⇒ cor_at(C1, ms(I, C1, C2))
     ∨ ctxt(C3, C1) ∧ cor_at(C3, ms(I, C1, C2)).                    % (17)

% Once corruption occurs, track it in later events.
prec(cor(I1, C), ms(I2, C, C1))
  ⇒ cor_at(C, ms(I2, C, C1)) ∨ rep_at(C, ms(I2, C, C1)).           % (18)
prec(cor(I1, C), ms(I2, C1, C))
  ⇒ cor_at(C, ms(I2, C1, C)) ∨ rep_at(C, ms(I2, C1, C)).           % (19)

% Lemma 2:
cor_at(C, E1) ⇒ prec(E0, E1) ∧ E0 = cor(I, C) ∧ lastAdvEv(E0, E1) ∧
  relevant(C, E0).                                                  % (20)

% Lemma 3:
cor_at(C, E1) ∧ rep_at(C, E2) ∧ prec(E1, E2)
  ⇒ prec(E1, E3) ∧ prec(E3, E2) ∧ E3 = rep(I, C)
     ∧ relevant(C, E3).                                             % (21)
rep_at(C, E1) ∧ cor_at(C, E2) ∧ prec(E1, E2)
  ⇒ prec(E1, E3) ∧ prec(E3, E2) ∧ E3 = cor(I, C)
     ∧ relevant(C, E3).                                             % (22)
prec(cor(I, C), E2) ∧ rep_at(C, E2)
  ⇒ prec(cor(I, C), E3) ∧ prec(E3, E2) ∧ E3 = rep(I1, C)
     ∧ relevant(C, E3).                                             % (23)
prec(rep(I, C), E2) ∧ cor_at(C, E2)
  ⇒ prec(rep(I, C), E3) ∧ prec(E3, E2) ∧ E3 = cor(I1, C)
     ∧ relevant(C, E3).                                             % (24)

%%%%%%%%%%%%%%%%
% Initial model %
%%%%%%%%%%%%%%%%

true
  ⇒ prec(attStart(i2), ms(i6, vc, sys))
       ∧ cor_at(sys, ms(i6, vc, sys)) ∧ ms_ok(ms(i6, vc, sys))
       ∧ ctxt(ker, vc).                                             % (25)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Facts about ctxt in our system %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ctxt(C, rtm) ⇒ false.                                    % (26)
ctxt(C, a1) ⇒ false.                                     % (27)
ctxt(C, a2) ⇒ false.                                     % (28)
ctxt(C, ker) ⇒ false.                                    % (29)
ctxt(C, sys) ⇒ false.                                    % (30)
ctxt(C, vc) ⇒ C = ker.                                   % (31)

%%%%%%%%%%%%%%%%%
% Recent or Deep %
%%%%%%%%%%%%%%%%%

prec(attStart(I), cor(I1, vc)) ⇒ recent(vc).
prec(attStart(I), cor(I1, ker)) ⇒ recent(ker).
prec(attStart(I), cor(I1, sys)) ⇒ recent(sys).
relevant(a1, cor(I, a1)) ⇒ deep(a1).
relevant(a2, cor(I, a2)) ⇒ deep(a2).
```

References

1. Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 2001.
2. John D. Ramsdell, Paul D. Rowe, Perry Alexander, Sarah C. Helble, Peter Loscocco, J. Aaron Pendergrass, and Adam Petz. Orchestrating layered attestations. In *Principles of Security and Trust (POST)*, 2019.
3. Paul D. Rowe. Bundling evidence for layered attestation. In *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016, Vienna, Austria, August 29-30, 2016, Proceedings*, pages 119–139, 2016.
4. Paul D. Rowe. Confining adversary actions via measurement. In *Graphical Models for Security - Third International Workshop, GraMSec 2016, Lisbon, Portugal, June 27, 2016, Revised Selected Papers*, pages 150–166, 2016.