Hyperproperties*

Michael R. Clarkson Fred B. Schneider

{clarkson,fbs}@cs.cornell.edu
Department of Computer Science
Cornell University

Abstract

Properties, which have long been used for reasoning about systems, are sets of traces. Hyperproperties, introduced here, are sets of properties. Hyperproperties can express security policies, such as secure information flow, that properties cannot. Safety and liveness are generalized to hyperproperties, and every hyperproperty is shown to be the intersection of a safety hyperproperty and a liveness hyperproperty. A verification technique for safety hyperproperties is given and is shown to generalize prior techniques for verifying secure information flow. Refinement is shown to be valid for safety hyperproperties. A topological characterization of hyperproperties is given.

1 Introduction

Important classes of *security policies* cannot be expressed using what have been termed *properties* [1, 26, 45, 12, 36, 42], sets of execution traces [20] for which membership of a trace depends on the trace alone and not on which other traces are in the property. For example, *noninterference* [13] is a confidentiality policy that stipulates commands executed on behalf of users holding high clearances have no effect on system behavior observed by users with only low clearances. It is not a property, because whether some given trace is allowed depends on whether another trace (obtained by deleting command executions by high users) is allowed. As a second example, stipulating a bound on average response time over all executions is an availability policy that cannot be specified as a property, because the acceptability of delays in any given execution depends on the magnitude of delays in all other executions.

Methods for specifying and reasoning about properties that a system satisfies are well understood [38, 22, 21]. It has been shown that every property is the intersection of a safety property and a liveness property,[1] where

- a *safety* property proscribes "bad things" and can be proved using an invariance argument, and
- a *liveness* property prescribes "good things" and can be proved using a well-foundedness argument.

Safety and liveness thus not only form an intuitively appealing fundamental basis from which all properties can be constructed, but they also are associated with specific verification methods. An analogous theory for security policies would be quite appealing. The fact that security policies also proscribe and prescribe behaviors of systems suggests that such a theory might exist.

This paper initiates the development of that theory by introducing *hyperproperties*, which are sets of properties (i.e., sets of sets of traces), and defining two interesting classes of hyperproperties: safety and liveness. We show:

- Hyperproperties can describe properties and, moreover, can describe security policies, such as noninterference and average response time, that properties cannot. Indeed, we have not been able to find requirements on system behavior that cannot be specified as a hyperproperty. Deterministic, nondeterministic, and probabilistic system models all can be handled using hyperproperties.
- Every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty. (Henceforth, we shorten these terms to *hypersafety* and *hyperliveness*.) Hypersafety and hyperliveness thus form a fundamental basis from which all hyperproperties can be constructed.
- The topological characterization of properties [4] can be generalized to characterize hyperproperties, and the result is equivalent to the *lower Vietoris* topology [44, 28, 39].

*Supported in part by AFOSR grant F9550-06-0019, National Science Foundation Grants 0430161 and CCF-0424422 (TRUST), an Intel Foundation PhD Fellowship, and a gift from Microsoft Corporation.

[1] Lamport [18] gave the first informal definitions for safety and liveness properties, appropriating the terms from Petri net theory, and he gave the first formal definition of safety [20]. Alpern and Schneider [4] gave the first formal definition of liveness and the proof that all properties are the intersection of safety and liveness properties; they later established the correspondence of safety to invariance and of liveness to well-foundedness [5].

978-0-7695-3182-3/08 $25.00 © 2008 IEEE
DOI 10.1109/CSF.2008.7

IEEE computer society

We have not obtained complete verification methods for hypersafety or for hyperliveness, but we have been able to generalize prior work on using invariance arguments to verify information-flow policies [7, 42]. Our generalization is applicable to a class of hyperproperties we introduce called *k-safety*.

The theory we have developed is also able to shed light on the problematic status of refinement for security policies. Refinement never invalidates a property but can invalidate a hyperproperty: Consider a system π that nondeterministically chooses to output 0, 1, or the value of a secret bit h. System π satisfies the security policy "The possible output values are independent of the values of secrets." But one refinement of π is the system that always outputs h, and this does not satisfy the security policy. Previous work has identified certain policies [25] and composition operators [26] that are suitable for use with refinement; we show in this paper that satisfaction of safety hyperproperties is preserved under refinement of nondeterminism, yielding an entire class of security policies to which refinement is applicable.

We proceed as follows. Hyperproperties, hypersafety, k-safety, and hyperliveness are defined and explored in Sections 2, 3, 4, and 5, respectively. Section 6 presents the hyperproperty intersection theorem, topology is addressed in Section 7, and Section 8 concludes. A guide to notation is provided in Appendix A, the formal details of some of our longer examples of hyperproperties are given in Appendix B, and all proofs appear in the accompanying technical report [11].

2 Hyperproperties

Many formalisms exist for modeling systems. We model system execution with traces, where a *trace* is a sequence of states; by employing rich enough notions of state, this model is sufficiently general to encode many other representations of executions.[2] The structure of a state is not important in the following definitions, so we leave Σ, the set of states, abstract. However, the structure of a state is important for real examples, so we introduce predicates and functions, on states and on traces, as needed—e.g., for events, timing, and probability.

Traces may be finite or infinite and are categorized into the following sets:

$$\begin{aligned} \Psi_{\mathsf{fin}} &\triangleq \Sigma^* \\ \Psi_{\mathsf{inf}} &\triangleq \Sigma^\omega \\ \Psi &\triangleq \Psi_{\mathsf{fin}} \cup \Psi_{\mathsf{inf}}. \end{aligned}$$

[2] Appendix B discusses how to model a labeled transition system as a set of traces, without losing information about the nondeterministic structure of the system. We leave the investigation of the meaning of hyperproperties in other models [46] as future work.

For trace $t = s_0 s_1 \ldots$ and index i, define the following indexing notation:

$$\begin{aligned} t[i] &\triangleq s_i \\ t[..i] &\triangleq s_0 s_1 \ldots s_i \\ t[i..] &\triangleq s_i s_{i+1} \ldots \end{aligned}$$

Concatenation of finite trace t and (finite or infinite) trace t' is denoted tt'. The empty trace is denoted ϵ.

A *system* is modeled by a non-empty set of infinite traces, called its *executions*. If a system execution terminates (and thus could be represented by a finite trace), we represent it as an infinite trace by infinitely stuttering the final state in the finite trace.

2.1 Properties

A *property* is a set of infinite traces. The set of all properties is

$$\mathsf{Prop} \triangleq \mathcal{P}(\Psi_{\mathsf{inf}}),$$

where $\mathcal{P}$ denotes powerset. A set T of traces satisfies a property P, denoted $T \models P$, iff all the traces of T are in P:

$$T \models P \triangleq T \subseteq P.$$

Some security policies are expressible as properties. For example, consider the policy, "The system may not write to the network after reading from a file." Formally, this is the set of traces

$$\begin{aligned} NRW \triangleq \{t \in \Psi_{\mathsf{inf}} \mid \neg(\exists\, i, j \in \mathbb{N} : i < j \\ \wedge\ isFileRead(t[i]) \\ \wedge\ isNetworkWrite(t[j]))\}, \end{aligned} \tag{2.1}$$

where $isFileRead$ and $isNetworkWrite$ are predicates on states. Similarly, *access control* is a property requiring every operation to be consistent with its requestor's rights:

$$\begin{aligned} AC \triangleq \{t \in \Psi_{\mathsf{inf}} \mid (\forall\, i \in \mathbb{N} : \\ rights(t[i]) \subseteq acm(t[i-1])[subj(t[i]), \\ obj(t[i]))]\}. \end{aligned} \tag{2.2}$$

Function $acm(s)$ yields the access control matrix in state s. Function $subj(s)$ yields the subject who requested the operation that led to state s, function $obj(s)$ yields the object involved in that operation, and function $rights(s)$ yields the right(s) necessary for the operation to be allowed.

As another example, *guaranteed service* is a property requiring that every request for service is eventually satisfied:

$$\begin{aligned} GS \triangleq \{t \in \Psi_{\mathsf{inf}} \mid (\forall\, i \in \mathbb{N} : \\ isReq(t[i]) \implies \\ (\exists\, j > i : isRespToReq(t[j], t[i]))\}. \end{aligned} \tag{2.3}$$

Predicate $isReq(s)$ identifies whether a request is initiated in state s, and predicate $isRespToReq(s', s)$ identifies whether state s' completes the response to the request initiated in state s.

2.2 Hyperproperties

A *hyperproperty* is a set of sets of infinite traces, or equivalently, a set of properties. The set of all hyperproperties is

$$\begin{aligned} \mathsf{HP} &\triangleq \mathcal{P}(\mathcal{P}(\Psi_{\mathsf{inf}})) \\ &= \mathcal{P}(\mathsf{Prop}). \end{aligned}$$

The interpretation of a hyperproperty as a security policy is that the hyperproperty specifies exactly the systems allowed by that policy. Each property in a hyperproperty is an allowed system, specifying exactly which executions are possible for that system. Thus a set T of traces satisfies hyperproperty $\boldsymbol{H}$, denoted $T \models \boldsymbol{H}$, iff T is in $\boldsymbol{H}$:

$$T \models \boldsymbol{H} \triangleq T \in \boldsymbol{H}.$$

Note the use of bold type to denote hyperproperties and sets of hyperproperties. See Appendix A for a guide to our other typographical conventions and notation.

Given a property P, there is a unique hyperproperty, which we denote $[P]$, that expresses the same policy as P. We call this hyperproperty the *lift* of P. For P and $[P]$ to express the same policy, they must be satisfied by the same sets of traces. Thus we can derive a definition of $[P]$:

$$\begin{aligned} & (\forall T \in \mathsf{Prop} : T \models P \equiv T \models [P]) \\ = \; & (\forall T \in \mathsf{Prop} : T \subseteq P \equiv T \in [P]) \\ = \; & [P] = \{T \in \mathsf{Prop} \mid T \subseteq P\} \\ = \; & [P] = \mathcal{P}(P). \end{aligned}$$

Consequently, $[P] \triangleq \mathcal{P}(P)$.

2.3 Hyperproperties in Action

Properties are satisfied by traces, whereas hyperproperties are satisfied by sets of traces. This additional level of sets means that hyperproperties can be more expressive than properties. We explore this added expressivity with some examples.

Information flow. Information-flow security policies express requirements on what information may be learned by users of a system. Users interact with systems by providing inputs and observing outputs. To model this interaction, define function $ev(s)$ as the input or output event, if any, that occurs when a system transitions to state s. Assume that at most one event, input or output, can occur at each transition. Extend this notation to $ev(t)$, denoting the sequence of events resulting from application of $ev(\cdot)$ to each state in trace t.[3] We further assume that each user of a system is cleared at confidentiality level L, representing *low* (i.e., public) information, or H, representing *high* (i.e., secret) information, and that each event is labeled with one of these confidentiality levels. Define $ev_L(t)$ to be the subsequence of low events contained within $ev(t)$, and $ev_{Hin}(t)$ to be the subsequence of high input events contained within $ev(t)$.

Noninterference, as defined by Goguen and Meseguer [13], requires that commands issued by users holding high clearances be removable without affecting observations of users holding low clearances. Treating commands as inputs and observations as outputs, we model this policy as a hyperproperty requiring a system to contain, for any trace t, a trace t' that has no high inputs yet has the same low events as t:

$$\begin{aligned} \boldsymbol{GMNI} \triangleq \; & \{T \in \mathsf{Prop} \mid T \in \boldsymbol{GMSys} \\ & \implies (\forall t \in T : (\exists t' \in T : \\ & ev_{Hin}(t') = \epsilon \\ & \wedge \; ev_L(t) = ev_L(t')))\}. \end{aligned} \tag{2.4}$$

Antecedent $T \in \boldsymbol{GMSys}$ expresses the requirement that T be a system satisfying the assumptions made by Goguen and Meseguer's formalization: T must be deterministic, and total with respect to inputs. We omit formalizing these requirements as hyperproperties.

Generalized noninterference [24] generalizes Goguen and Meseguer's definition of noninterference to nondeterministic systems. McLean's formulation [26] of generalized noninterference requires a system to contain, for any traces t_1 and t_2, an interleaved trace t_3 whose high inputs are the same as t_1 and whose low events are the same as t_2. This is a hyperproperty:

$$\begin{aligned} \boldsymbol{GNI} \triangleq \; & \{T \in \mathsf{Prop} \mid (\forall t_1, t_2 \in T : \\ & (\exists t_3 \in T : ev_{Hin}(t_3) = ev_{Hin}(t_1) \\ & \wedge \; ev_L(t_3) = ev_L(t_2)))\}. \end{aligned} \tag{2.5}$$

Observational determinism [35, 47] requires a system to appear to a low user as a deterministic function of only the low inputs. Thus, it is a hyperproperty requiring that if any two traces have the same first $j-1$ low events, then these traces must have equivalent j^{th} low events:

$$\begin{aligned} \boldsymbol{OD} \triangleq \; & \{T \in \mathsf{Prop} \mid (\forall t_1, t_2 \in T, j \in \mathbb{N} : \\ & ev_L(t_1)[..j-1] = ev_L(t_2)[..j-1] \\ & \implies ev_L(t_1)[j] \approx_{in} ev_L(t_2)[j] \\ & \vee \; ev_L(t_1)[j] \approx_{out} ev_L(t_2)[j])\}. \end{aligned} \tag{2.6}$$

[3] Depending on the nature of events in the particular system that is being modeled, it may be appropriate for $ev(t)$ to eliminate stuttering of events.

Here we have extended trace indexing notation to apply to sequences of events. Events l_1 and l_2 are low input equivalent, denoted $l_1 \approx_{in} l_2$, iff they are both low input events (although the value input need not be the same in the two events). In contrast, events l_1 and l_2 are low output equivalent, denoted $l_1 \approx_{out} l_2$, iff they are both low output events of the same value.

Bisimulation-based definitions of information-flow security policies can also be formulated as hyperproperties.[4] We give an example in Appendix B by formulating, as hyperproperty ***BCNI***, Boudol and Castellani's [8] bisimulation-based definition of noninterference.

All information-flow security policies we investigated were found to be hyperproperties—not properties. This is suggestive, but any stronger statement about the connection between information flow and hyperproperties would require a formal definition of information flow policies, and none is universally accepted. We believe, however, that information flow is intrinsically tied to correlations between (not within) executions. Hyperproperties are sufficiently expressive to formulate such correlations, whereas properties are not. In particular, ***GMNI*** is not a property, as argued in Section 1, ***GNI*** is not a property because the presence of any two traces in the system necessitates the presence of a third trace, and ***OD*** is not a property because whether some trace is allowed depends on the low events appearing in all other traces of the system.

Service level agreements. A *service level agreement* (SLA) specifies acceptable performance of a system. Such specifications commonly use statistics, including:

- *average response time*, the average time that elapses between a request and a response;
- *time service factor*, the percentage of requests that are serviced within a specified time; and
- *percentage uptime*, the percentage of time during which the system is available to accept and service requests.

These statistics can be used to define policies with respect to each individual execution of a system or across all executions of a system. In the former case, the SLA would be a property. For example, the policy "The average response time *in each execution* is less than 1 second" might not be satisfied by a system if there are executions in which some response times are much greater than 1 second. Yet if these executions are rare, then the system might still satisfy the policy "The average response time *over all executions* is less than 1 second." This latter SLA is not a property, but it is a hyperproperty and can be stated formally as

$$\textbf{\textit{RT}} \triangleq \{T \in \mathsf{Prop} \mid mean(\bigcup_{t \in T} respTime(t)) \leq 1\}. \tag{2.7}$$

Function $mean(X)$ denotes the mean of a set X of real numbers, and function $respTime(t)$ denotes the set of response times (in seconds) from request/response events in trace t.[5] Policies derived from the other SLA statistics above can similarly be expressed as hyperproperties.

Refinement. One of the key differences between properties and hyperproperties is how they behave with respect to *refinement of nondeterminism*—removing traces from a system's set of executions. A system S is *refined by* system S' iff $S \supseteq S'$. By definition, whenever a system satisfies a property, any refinement of the system also satisfies the property. Thus, properties are *refinement-closed*:

$$S \models P \ \wedge \ S \supseteq S' \quad \Longrightarrow \quad S' \models P.$$

A hyperproperty is refinement-closed if whenever a system satisfies the hyperproperty, any refinement of the system also satisfies the hyperproperty. Define RC to be the set of refinement-closed hyperproperties. Hyperproperties resulting from lifted properties are refinement-closed:

$$S \models [P] \ \wedge \ S \supseteq S' \quad \Longrightarrow \quad S' \models [P].$$

However, hyperproperties in general are not refinement-closed. System π (Section 1) illustrates this fact.

Beyond hyperproperties? We introduced another level of sets when generalizing properties to hyperproperties, and in doing so we gained expressive power for specifying policies on systems. Thus, it is natural to ask whether introducing yet one more level of sets might also be useful. We believe it is not. Suppose, for sake of contradiction, that some set $\mathcal{H}$ of hyperproperties (i.e., $\mathcal{H}$ is a set of sets of sets of traces) was more expressive than any hyperproperty. Whatever the definition of satisfaction, $\mathcal{H}$ must either be satisfied or not satisfied by any system S. So consider set $\boldsymbol{H}$ of all systems that satisfies $\mathcal{H}$. But $\boldsymbol{H}$ is a hyperproperty (since it is a set of sets of traces), and $\boldsymbol{H}$ is equivalent to $\mathcal{H}$, so $\mathcal{H}$ is not more expressive than any hyperproperty.

[4] Since hyperproperties are trace-based, this might at first seem to contradict results, such as Focardi and Gorrieri's [12], stating that bisimulation-based definitions are stronger (i.e., a finer equivalence) than trace-based definitions. However, by employing a richer notion of state [38, §1.3] in traces than Focardi and Gorrieri do, our hyperproperties are able to express bisimulations.

[5] For $mean(\cdot)$ to be well-defined, it suffices that there be only a finite number of requests in T and that every request is serviced in finite time. The formulation of ***RT*** assumes all traces are equally likely. Modeling the case where some traces are more likely than others requires a probability measure on sets of traces. Obtaining such a measure is discussed in Section 6.

Another way to rationalize adding a level of sets would be to consider policies on sets of systems. For example, a policy might require that a set of systems exhibit sufficient diversity [34], meaning the systems all implement the same functionality but differ in their implementation details. This policy can be modeled as a hyperproperty on a single system that is a product[6] of all the systems in the set. More generally, a policy on a sequence $\boldsymbol{\mathcal{S}}$ of systems might be modeled as a set $\boldsymbol{\mathcal{H}}$ of sequences of hyperproperties. Again, by taking the product of each element of $\boldsymbol{\mathcal{H}}$, we obtain an equivalent hyperproperty.

The above conclusions will not surprise students of mathematical logic [27]. In first-order logic, variables range over individual elements of some universe; in second-order logic, variables may also range over subsets of the universe. If the universe is the set Ψ_{inf} of traces, then properties are first-order predicates on traces, and hyperproperties are second-order predicates on traces. Second-order logic is more expressive than first-order logic [43, §2.2], just as hyperproperties are more expressive than properties. Further, any higher-order logic (which would have variables ranging over sets (of sets of…sets) of subsets of the universe) is reducible to second-order logic [43, §4.3], just as we have reduced extra levels of sets, above, to hyperproperties. We leave further investigation of this connection as future work. One interesting avenue to explore would be whether the full power of second-order logic is necessary to express hyperproperties of interest. This has ramifications for verification of hyperproperties, because although full second-order logic cannot be effectively and completely axiomatized, fragments of it can be [43, §2.3].

3 Hypersafety

According to Alpern and Schneider [4], the "bad thing" in a safety property must be both

- *finitely observable*, meaning its occurrence can be detected in finite time, and
- *irremediable*, so its occurrence can never be remediated by future events.

For example, no-read-then-write NRW (2.1) and access control AC (2.2) are both safety. The bad thing for NRW is a finite trace in which a network write occurs after a file read. This bad thing is finitely observable, because the write can be detected in some finite prefix of the trace, and irremediable, because the network write can never be undone. For AC, the bad thing is similarly a finite trace in which an operation is performed without appropriate rights.

[6]The *product* of systems T_1 and T_2 is the system comprising traces over pairs of states, defined as: $T_1 \times T_2 \triangleq \{(t_1[0], t_2[0])(t_1[1], t_1[2]) \ldots \mid t_1 \in T_1 \ \wedge \ t_2 \in T_2\}$. Generalizing, the product of a set of n systems comprises traces over n-tuples of states.

A bad thing is a finite trace that cannot be a prefix of any execution satisfying the safety property. A finite trace t is a *prefix* of a (finite or infinite) trace t', denoted $t \leq t'$, iff $t' = tt''$ for some $t'' \in \Psi$.

Safety property. A property S is a *safety property* [4] iff

$$(\forall\, t \in \Psi_{\mathsf{inf}} : t \notin S \implies (\exists\, m \in \Psi_{\mathsf{fin}} : m \leq t \wedge (\forall\, t' \in \Psi_{\mathsf{inf}} : m \leq t' \implies t' \notin S))). \quad \square$$

Define SP to be the set of all safety properties. Notice that SP is itself a hyperproperty.

We generalize safety to hypersafety by generalizing the bad thing from a finite trace to a finite[7] set of finite traces. Define Obs to be the set of such *observations*:

$$\mathsf{Obs} \triangleq \mathcal{P}^{fin}(\Psi_{\mathsf{fin}}),$$

where $\mathcal{P}^{fin}(X)$ denotes the set of all finite subsets of set X. Prefix $\leq$ on sets of traces is defined as:[8]

$$T \leq T' \triangleq (\forall\, t \in T : (\exists\, t' \in T' : t \leq t')).$$

Note that this definition allows T' to contain new traces that have no prefix in T.

Safety hyperproperty. A hyperproperty $\boldsymbol{S}$ is a *safety hyperproperty* (equivalently, is *hypersafety*) iff

$$(\forall\, T \in \mathsf{Prop} : T \notin \boldsymbol{S} \implies (\exists\, M \in \mathsf{Obs} : M \leq T \wedge (\forall\, T' \in \mathsf{Prop} : M \leq T' \implies T' \notin \boldsymbol{S}))). \quad \square$$

The definition of hypersafety parallels the definition of safety—the only change is that the domains involved now include an extra level of sets. Define SHP to be the set of all safety hyperproperties.

Some consequences of the definition of hypersafety are:

- Goguen and Meseguer's noninterference ***GMNI*** (2.4) is hypersafety. The bad thing is a pair (t, t') of traces where t' contains no high inputs and contains the same low inputs as t, yet t and t' have different low outputs.
- Observational determinism ***OD*** (2.6) is hypersafety. The bad thing is a pair of traces whose first $j-1$ low events are the same, yet whose j^{th} events are different low outputs.

[7]Infinite sets might at first seem an attractive alternative, and many of the results in the rest of this paper would still hold. However, the topological characterization given in Section 7 (specifically, Propositions 5 and 6) would be sacrificed.

[8]Other definitions of prefix are possible, but inconsistent with our notion of observation. This definition coincides with the ordering of the lower (or Hoare) powerdomain on traces. We discuss this in Section 7.

- Safety properties lift to safety hyperproperties.

  Proposition 1. $(\forall S \in \mathsf{Prop} : S \in \mathsf{SP} \iff [S] \in \mathsf{SHP})$

- Set SP of all safety properties is not a safety hyperproperty: There is no bad thing that prevents an arbitrary property from being extended to some safety property.

Refinement of hypersafety. All safety hyperproperties are refinement-closed. Intuitively, this is because if a bad thing excludes property T from membership in some safety hyperproperty, then any property of which T is a refinement would also contain the same bad thing.

Theorem 1. $\mathsf{SHP} \subset \mathsf{RC}$

By this theorem, any information-flow security policy that is not refinement-closed cannot be hypersafety. For example, generalized noninterference ***GNI*** (2.5) is not hypersafety, because it is not refinement-closed: A system containing traces t_1 and t_2, yet not containing the interleaved trace t_3 required by the definition of ***GNI***, may be extended to a system containing t_3.

Relational hyperproperties. A program might be modeled as a system with a single action, which transitions from the *input state* (the initial state in the execution) to the *output state* (the final state in the execution) with no other observable states.[9] Define a *relational* hyperproperty as a hyperproperty on traces with such a single action. The first state in each trace is the initial state, the second state is the final state, and the second state is infinitely stuttered to produce an infinite trace. Define Ψ_R to be the set of such traces, and define RHP to be $\mathcal{P}(\mathcal{P}(\Psi_R))$, the set of all relational hyperproperties.

Relational hyperproperties facilitate the definition of an information-flow security policy that is commonly used in language-based security [37]. This policy, which we call *relational noninterference*, requires execution of a program π to maintain the equivalence of states to a low observer. That is, if s_i' is the output state resulting from executing π with input state s_i, and s_1 and s_2 are low-equivalent, then s_1' must be low-equivalent to s_2'. In our formalism, relational noninterference can be defined as:

$$\begin{aligned} \textbf{\textit{RNI}} \triangleq \{T \subseteq \Psi_R \mid (\forall\, t_1, t_2 \in T : \\ ev_L(t_1)[0] = ev_L(t_2)[0] \\ \implies ev_L(t_1)[1] = ev_L(t_2)[1])\}. \end{aligned} \tag{3.1}$$

Inspecting this definition reveals it is a refinement of observational determinism ***OD*** (2.6) where $j = 1$. Since ***OD*** is hypersafety, ***RNI*** is also hypersafety.

[9] Since some programs do not terminate on some inputs, a special output state might be added to denote nontermination.

4 Beyond 2-Safety

Recent work gives system transformations that reduce verifying secure information flow to verifying a property of some transformed system. (Recall that secure information flow is a hyperproperty but not a property.) Pottier and Simonet [33] develop a type system for verifying secure information flow based on simultaneous reasoning about two executions of a program. Darvas et al. [3] show that secure information flow can be expressed in dynamic logic. Barthe et al. [7] give an equivalent formulation for Hoare logic and temporal logic, based on a self-composition construction.

Define the *sequential self-composition of* P as the program $P; P'$, where P' denotes program P, but with every variable renamed to a primed variable—e.g., variable x is renamed to x'. Then, one way to verify that (termination-insensitive) relational noninterference ***RNI*** (3.1) holds of program P is to establish the following property of transformed program $P; P'$:

> If for every low variable l, before execution $l = l'$ holds, then when execution terminates $l = l'$ still holds, no matter what the values of high variables were.

Barthe et al. generalize the self-composition operator from ; to any operator that satisfies certain conditions, and they note that parallel composition satisfies these conditions. They also relax the equality constraints in the above property to partial equivalence relations, obtaining a generalization of relational noninterference.

Terauchi and Aiken [42] further generalize the applicability of self-composition by showing that it can be used to verify any *2-safety* property, which they define informally as a "property that can be refuted by observing two finite traces;" their formal definition is very similar to a relational hyperproperty.

Using hyperproperties, we can show that the above results are a special case of a more general theorem. Define a k-safety hyperproperty as a safety hyperproperty in which the bad thing never involves more than k traces.

***k*-safety hyperproperty.** A hyperproperty $\boldsymbol{S}$ is a *k-safety hyperproperty* (equivalently, is *k-safety*) iff

$$\begin{aligned} (\forall T \in \mathsf{Prop} : T \notin \boldsymbol{S} \implies (\exists M \in \mathsf{Obs} : \\ M \leq T \ \wedge\ |M| \leq k \ \wedge\ (\forall T' \in \mathsf{Prop} : \\ M \leq T' \implies T' \notin \boldsymbol{S}))). \end{aligned} \quad \square$$

This is the definition of hypersafety, with an added conjunct "$|M| \leq k$". Given a particular k, define $\mathsf{KSHP}(k)$ to be the set of all k-safety hyperproperties.

As an example of a k-safety hyperproperty for any k, consider a system that stores a secret by splitting it into

k shares. Suppose that an action of the system is to output share i. Then a hyperproperty of interest might be that the system cannot, across any of its executions, output all k shares (thereby outputting sufficient information for the secret to be reconstructed). We denote this k-safety hyperproperty as $\boldsymbol{SS}_k$.

Note that the 1-safety hyperproperties are the lifted safety properties,

$$\mathsf{KSHP}(1) \quad = \quad \{[S] \mid S \in \mathsf{SP}\},$$

since the bad thing for a safety property is a single trace. Thus "1-safety" and "safety" are synonymous.

The Terauchi and Aiken definition of 2-safety properties (which we now identify as $\mathsf{KSHP}(2)$, the 2-safety hyperproperties) is based on a relational model of program execution, so it is limited to expressing relational 2-safety hyperproperties. Relational noninterference ***RNI*** (3.1) is an example of such a hyperproperty. Our definition, based on a trace model of execution, is more general and allows us to conclude that Goguen and Meseguer's noninterference ***GMNI*** (2.4) and observational determinism ***OD*** (2.6), which are not relational, are also 2-safety hyperproperties.[10]

Define the *parallel self-composition of system* S as the product system $S \times S$ consisting of traces over $\Sigma \times \Sigma$:

$$S \times S \quad \triangleq \quad \{(t[0], t'[0])(t[1], t'[1]) \ldots \mid t \in S \ \wedge \ t' \in S\}.$$

Define the k-*product* of system S, denoted S^k, to be the k-fold parallel self-composition of S, comprising traces over Σ^k. Self-composition $S \times S$ is equivalent to 2-product S^2.

Previous work has shown how to reduce a 2-safety hyperproperty of system S to a related safety property of S^2. The following theorem generalizes that. Let Sys be the set of all systems. Then, for any system S, any k-safety hyperproperty of S can be reduced to a safety property of S^k.

Theorem 2. $(\forall\, S \in \mathsf{Sys}, \boldsymbol{K} \in \mathsf{KSHP}(k) \ : \ (\exists\, K \in \mathsf{SP} \ : \ S \models \boldsymbol{K} \iff S^k \models K))$

The proof of this theorem (in the accompanying technical report [11]) shows how to construct K from $\boldsymbol{K}$. Thus, Theorem 2 suggests a verification technique for k-safety, namely to reduce a k-safety hyperproperty to a safety property, then verify the safety property is satisfied by S^k using invariance arguments. Since invariance arguments are relatively complete for safety properties [5], Theorem 2 yields a relatively complete verification methodology for k-safety.

However, Theorem 2 does not provide the relatively complete verification procedure we seek for hypersafety, because there are safety hyperproperties that are not k-safety for any k. For example, consider the hyperproperty "a system cannot output all k shares of a secret for any k-secret sharing;" formally, this is

$$\boldsymbol{SS} \quad \triangleq \quad \bigcup_k \boldsymbol{SS}_k.$$

This is not k-safety for any k, yet it is hypersafety, since any property not contained in it violates some $\boldsymbol{SS}_k$.

[10] This conclusion resolves the conjecture of Terauchi and Aiken that (termination-sensitive) secure information flow over infinite traces is "2-liveness [*sic*]," for some definition of 2-liveness.

5 Hyperliveness

According to Alpern and Schneider [4], the "good thing" in a liveness property is

- *always possible*, no matter what has occurred so far, and
- *possibly infinite*, so it need not be a discrete event.

For example, guaranteed service GS (2.3) is a liveness property in which the good thing is the eventual response to a request. This good thing is always possible because a response can always be appended to any finite trace containing a request, and it is not infinite because the response is a discrete event.

Liveness property. Property L is a *liveness property* [4] iff

$$(\forall\, t \in \Psi_{\mathsf{fin}} \ : \ (\exists\, t' \in \Psi_{\mathsf{inf}} \ : \ t \leq t' \ \wedge \ t' \in L)). \quad \square$$

Define LP to be the set of all liveness properties. Not surprisingly, LP is itself a hyperproperty.

Just as with hypersafety, we generalize liveness to hyperliveness by generalizing a finite trace to a finite set of finite traces. The definition of hyperliveness is essentially the same as the definition of liveness, except for an additional level of sets.

Liveness hyperproperty. Hyperproperty $\boldsymbol{L}$ is a *liveness hyperproperty* (equivalently, is *hyperliveness*) iff

$$(\forall\, T \in \mathsf{Obs} \ : \ (\exists\, T' \in \mathsf{Prop} \ : \ T \leq T' \ \wedge \ T' \in \boldsymbol{L})). \ \square$$

Define LHP to be the set of all liveness hyperproperties.

Some consequences of the definition of hyperliveness are:

- Average response time ***RT*** (2.7) is not liveness but it is hyperliveness: the good thing is that the average response time is low enough. If this policy were approximated by limiting the maximum (rather than mean) response time in each execution, the hyperproperty would instead be a lifted safety property.
- The only hyperproperty that is both hypersafety and hyperliveness is ***true***, where ***true*** $\triangleq$ Prop, the maximal hyperproperty with respect to the subset relation. (The minimal hyperproperty ***false***, where ***false*** $\triangleq \{\emptyset\}$, is hypersafety but not hyperliveness.)

- Liveness properties lift to liveness hyperproperties.

  Proposition 2. $(\forall\, L \in \mathsf{Prop} \;:\; L \in \mathsf{LP} \iff [L] \in \mathsf{LHP})$

- Set LP of all liveness properties is a liveness hyperproperty: Every observation can be extended to any liveness property.

- Similarly, set SP of all safety properties is a liveness hyperproperty: Every observation can be extended to some safety property.

Possibilistic information flow. Some information-flow security policies, such as observational determinism ***OD*** (2.6) and relational noninterference ***RNI*** (3.1), restrict nondeterminism of a system from being publicly observable. However, it could be useful to have observable nondeterminism. First, systems might exhibit nondeterminism due to scheduling. For example, if the scheduler cannot be influenced by secret information (i.e., the scheduler does not serve as a covert timing channel), then it is reasonable to allow the scheduler to behave nondeterministically. Second, nondeterminism is a useful modeling abstraction when dealing with probabilistic systems (which we consider in more detail in Section 6). When the exact probabilities for a system are unknown, they can be abstracted by nondeterminism. For at least these reasons, there has been a history of research on *possibilistic* information-flow security policies, beginning with nondeducibility [41] and generalized noninterference [24]. Such policies are founded on the intuition that low observers of a system should gain little from their observations. Typically, these policies require that every low observation is consistent with some large set of possible high behaviors.

McLean [26] argues that every possibilistic information-flow security policy is expressible as a *selective interleaving function*. Such functions, given two executions of a system, specify another trace that must also be an execution of the system, as did the definition of generalized noninterference ***GNI*** (2.5). McLean shows that possibilistic information-flow policies can be expressed as closure with respect to selective interleaving functions. Mantel [23] generalizes from these functions to *closure operators*, which extend a set S of executions to a set S' such that $S \subseteq S'$. Mantel argues that every possibilistic information-flow policy can be expressed as a closure operator.

Given a closure operator Cl that expresses a possibilistic information-flow policy, the hyperproperty $\boldsymbol{P}_{Cl}$ induced by Cl is:

$$\boldsymbol{P}_{Cl} \;\triangleq\; \{Cl(T) \mid T \in \mathsf{Prop}\}.$$

Define the set PIF of all such hyperproperties to be $\bigcup_{Cl} \boldsymbol{P}_{Cl}$. It is now easy to see that these are liveness hyperproperties: any observation T can be extended to its closure.

Proposition 3. $\mathsf{PIF} \subset \mathsf{LHP}$

Possibilistic information-flow policies, other than ***true***, are therefore never hypersafety. Another way to reach this conclusion is to observe that closure operators sometimes yield hyperproperties that are not refinement-closed—yet, by Theorem 1, every safety hyperproperty is refinement-closed.

Temporal logics. Consider the hyperproperty "for every initial state, there is some terminating trace, though not all traces need terminate," denoted as ***DT***. In branching-time temporal logic, ***DT*** could be expressed more precisely as

$$\boldsymbol{DT} \;\triangleq\; \Diamond\mathit{terminates}, \tag{5.1}$$

where $\mathit{terminates}$ is a state predicate and $\Diamond$ is the "not never" operator.[11] There is no liveness property equivalent to ***DT*** [19]; an approximation would be the liveness property that requires every trace to terminate. However, ***DT*** is hyperliveness because any finite trace can be extended to a set of executions, at least one of which terminates. This example suggests that hyperproperties can be models for branching-time temporal predicates whereas properties are limited to modeling linear-time temporal predicates.

6 Other Hyperproperties

Hyperproperties are not necessarily either hypersafety or hyperliveness. Consider a medical information system that must maintain the confidentiality of patient records and must also eventually notify patients whenever their records are accessed [6]. Assuming the confidentiality requirement is interpreted as observational determinism ***OD*** (2.6), this system must both prevent bad things (***OD***, which is hypersafety) as well as guarantee good things (eventual notification, which can be formulated as liveness). As another example, consider a proactive secret sharing system that must maintain and periodically refresh a secret. Maintaining the confidentiality of the secret can be formulated as hypersafety, and the eventual refresh of the secret shares can be formulated either as liveness (if every execution must eventually complete the refresh) or hyperliveness (if only some executions must complete). Both of these examples illustrate hyperproperties that are intersections of (hyper)safety and (hyper)liveness.

In fact, every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty. This generalizes the result of Alpern and Schneider [4] that every property is the intersection of a safety property and a liveness property.

Theorem 3. $(\forall\, \boldsymbol{P} \in \mathsf{HP} \;:\; (\exists\, \boldsymbol{S} \in \mathsf{SHP}, \boldsymbol{L} \in \mathsf{LHP} \;:\; \boldsymbol{P} = \boldsymbol{S} \cap \boldsymbol{L}))$

[11] Some temporal logics, such as CTL [9], express this formula as $\mathsf{EF}\mathit{terminates}$.

Probabilistic Hyperproperties. Although the formulation of systems and hyperproperties in Section 2 did not include probabilities, it is straightforward to incorporate them. A probabilistic system transitions from a state s to a state s' with probability $p(s, s')$.[12] Define $\mathsf{Pr}_{s,S}(T)$ to be the probability with which set T of finite traces is produced by system S with initial state s. This measure can be constructed from $p(\cdot, \cdot)$ [17, 31] and used in the definitions of hyperproperties.

In information-flow security, the original motivations for adding probability to system models were to address covert channels and to establish connections between information theory and information flow [29, 14, 15]. A security policy that emerged from this line of research was *probabilistic noninterference* [16, 31]. Intuitively, this policy requires that the probability of every low trace be the same for every low-equivalent initial state. A formulation of this as hyperproperty ***PNI*** appears in Appendix B. ***PNI*** is an example of a hyperproperty that is intrinsically neither hypersafety nor hyperliveness: If two low traces have differing probabilities in some observation, it may or may not be possible to extend the observation to make the probabilities equal. Because it is neither always possible nor always impossible to do so, ***PNI*** is neither hypersafety nor hyperliveness.

To measure *quantity of leakage from repeated experiments* in probabilistic programs, Clarkson et al. [10] use a probabilistic denotational semantics. This semantics can be used to define a system, and the traces of the system represent repeated executions of the program. The hyperproperty "the quantity of leakage over every series of experiments on deterministic program S is less than k bits" (denoted ***QL***) then can be shown to be hypersafety. For details, see Appendix B.

The *channel capacity* of a system is the rate (defined as a limit over all execution lengths) at which information flows through the system [15]. The hyperproperty "the channel capacity is k bits" (denoted ***CC***) can be shown to be hyperliveness. Intuitively, no matter what the rate is for some finite prefix of the system, the rate can changed to any arbitrary amount by an appropriate extension that conveys more or less information.

7 Topology

Topology enables an elegant characterization of the structure of hyperproperties. We begin by summarizing the topology of properties [40].

Consider an *observer* of an execution of a system, who is permitted to see each new state as it is produced by the system; otherwise, the system is opaque to the observer. The observer attempts to determine whether property P holds of the system. At any point in time, the observer has seen only a finite prefix of the (infinite) execution. Thus, the observer should declare that the system satisfies P, after observing finite trace t, only if all possible extensions of t will also satisfy P. Abramsky names such properties *finitely observable* [2].

As with the bad thing for a safety property, a finitely observable property must be detectable in finite time, and once detected, hold thereafter. Formally, O is a finitely observable property iff

$$(\forall\, t \in \Psi_{\mathsf{inf}} : t \in O \implies (\exists\, m \in \Psi_{\mathsf{fin}} : m \leq t \\ \wedge\ (\forall\, t' \in \Psi_{\mathsf{inf}} : m \leq t' \implies t' \in O))).$$

Define $\mathcal{O}$ to be the set of finitely observable properties. Finitely observable properties satisfy two closure conditions. First, if $O_1, \ldots, O_n$ are finitely observable, then $\bigcap_{i=1}^{n} O_i$ is also finitely observable. Second, if $\boldsymbol{O}$ is a (potentially infinite) set of finitely observable properties, then $\bigcup_{O \in \boldsymbol{O}} O$ is also finitely observable. Thus we say that $\mathcal{O}$ is closed under finite intersections and infinite unions.

Recall that a *topology* on a set S is a set $\mathcal{T} \subseteq \mathcal{P}(S)$ such that $\mathcal{T}$ is closed under finite intersections and infinite unions. The elements of $\mathcal{T}$ are called *open sets*. Because $\mathcal{O}$ satisfies these requirements, it is a topology on Ψ_{inf}. We call $\mathcal{O}$ the *Plotkin* topology after its inventor.

A convenient way to characterize a topology is to define a *base* or a *subbase* for the topology. A base of topology $\mathcal{T}$ is a set $\mathcal{B} \subseteq \mathcal{T}$ such that every open set is a (potentially infinite) union of elements of $\mathcal{B}$. A subbase is a set $\mathcal{A} \subseteq \mathcal{T}$ such that the collection of finite intersections of $\mathcal{A}$ is a base for $\mathcal{T}$. A base (also a subbase) of the Plotkin topology is

$$\mathcal{O}^B \triangleq \{\uparrow t \mid t \in \Psi_{\mathsf{fin}}\},$$

where $\uparrow t \triangleq \{t' \in \Psi_{\mathsf{inf}} \mid t \leq t'\}$ is the *completion* of a finite trace t. When $t \leq t'$ we say that t' *extends* t. The completion of t is thus the set of all infinite extensions of t. The open sets $\mathcal{O}$ of the Plotkin topology are the sets in the closure of $\mathcal{O}^B$ under infinite unions.

Alpern and Schneider [4] established that safety properties correspond to closed sets, and liveness properties correspond to dense sets, in the Plotkin topology. A *closed* set is the complement of an open set, and a set that is *dense in* $\mathcal{T}$ intersects every non-empty open set in $\mathcal{T}$. (Hereafter, we shorten *dense in* $\mathcal{T}$ to *dense*.) Intuitively, non-membership in a closed set is finitely observable, with a finite trace constituting the bad thing for a safety property. And any finite observation can be extended to be in a dense set, constituting a good thing, so that dense set is live.

We want to construct a topology on sets of traces that extends this correspondence to hyperproperties. The most important step is generalizing the notion of finite observability from properties to hyperproperties. Section 3 already

[12] If this probability is dependent on the history of execution, then the implementation must have available, in each state, enough information to reconstruct that history.

did this in generalizing a finite trace to a finite set of finite traces—i.e., an observation. The observer, as before, sees the system produce each new state in the execution. However, the observer may now reset the system at any time, causing it to begin a new execution. At any finite point in time, the observer has now collected a finite set of finite (thus partial) executions.[13] An observation is thus an element of Obs, as defined in Section 3.

An extension of an observation should allow the observer to perform additional resets of the system, yielding a larger set of traces. An extension should also allow each execution to proceed longer, yielding longer traces. So an extension corresponds to trace set prefix $\leq$ as defined in Section 3. The completion of observation $M \in \mathsf{Obs}$ is

$$\uparrow M \triangleq \{T \in \mathsf{Prop} \mid M \leq T\}.$$

We can now define our topology on sets of traces in terms of its subbase:

$$\mathcal{O}^{SB} \triangleq \{\uparrow M \mid M \in \mathsf{Obs}\}.$$

The base $\mathcal{O}^B$ of our topology is then $\mathcal{O}^{SB}$ closed under finite intersections. The base and subbase turn out to be the same sets.

Proposition 4. $\mathcal{O}^B = \mathcal{O}^{SB}$

Finally, the topology $\mathcal{O}$ is $\mathcal{O}^B$ closed under infinite unions. Thus, open sets are unions of completed observations.

Next, we provide the appropriate characterizations of safety and liveness for this topology. Define $\mathcal{C}$ to be the closed sets and $\mathcal{D}$ to be the dense sets.

Proposition 5. $\mathsf{SHP} = \mathcal{C}$

Proposition 6. $\mathsf{LHP} = \mathcal{D}$

Thus, just as safety and liveness correspond to closed and dense sets in the Plotkin topology, hypersafety and hyperliveness correspond to closed and dense sets in our generalization of that topology.

The topology we developed here is actually equivalent to well-known topology. The *Vietoris* (or *finite* or *convex Vietoris*) topology is a standard construction of a topology on sets out of an underlying topology [44, 28]. This construction can be decomposed into the *lower Vietoris* and *upper Vietoris* constructions [39], which also yield topologies. Define $\mathfrak{V}_L(\mathcal{T})$ to be the lower Vietoris construction, which given topology $\mathcal{T}$ on space $\mathcal{X}$ produces a topology on $\mathcal{P}(\mathcal{X})$. Our underlying topology was on traces, and we constructed the lower Vietoris topology on sets of traces.

Theorem 4. $\mathcal{O} = \mathfrak{V}_L(\mathcal{O})$

[13] Equivalently, instead of allowing resets, the observer could run a finite collection of copies of the system. At any finite point in time, an observation can be made, comprising the set of traces the copies have produced.

This theorem yields another topological characterization of safety hyperproperties. The set of lifted safety properties, closed under infinite intersections and finite unions (denoted Ξ), is the set of safety hyperproperties.

Corollary 1. $\mathsf{SHP} = \Xi(\{[S] \mid S \in \mathsf{SP}\})$

Powerdomains. A *powerdomain* is a construction used to model the semantics of nondeterminism [32]. The definition of prefix $\leq$ over sets of traces in Section 3 suggests we are working with the *lower* (or *Hoare*) powerdomain on traces. Theorem 4 validates this, since the lower Vietoris topology corresponds to the lower powerdomain [39]. The two other standard powerdomain constructions, the *upper* (or Smyth) and *convex* (or Plotkin) powerdomains, similarly correspond to the upper and convex Vietoris topologies [39]. These two topologies use different open sets than the lower Vietoris topology we are using, changing the notions of observable and trace prefix: The upper construction makes all open sets refinable, whereas the convex construction makes the impossibility of the production of a state observable. Thus, neither the upper nor the convex constructions yield opens sets that are the finitely observable properties; the equivalence of closed sets and hypersafety consequently is lost. This means that the upper and convex powerdomains on traces are unsuitable for our purposes. It is possible that these powerdomains might nonetheless be useful for a different semantic domain than traces; we leave this as future work.

8 Concluding Remarks

Many examples of security policies have been classified as hyperproperties in this paper. Figure 1 summarizes this classification.

Although this paper formulates security policies with hyperproperties, security policies are typically formulated in terms of confidentiality, integrity, and availability. The relation between these two formulations is an open question, but we can offer some observations:

- Information-flow confidentiality is not a property, but it is a hyperproperty, and it can be hypersafety (e.g., observational determinism) or hyperliveness (e.g., generalized noninterference).

- Availability is sometimes hypersafety (maximum response time in any execution, which is also safety) and sometimes hyperliveness (mean response time over all executions).

- Integrity, which we have not discussed in this paper, also includes examples from both hypersafety and hyperliveness.

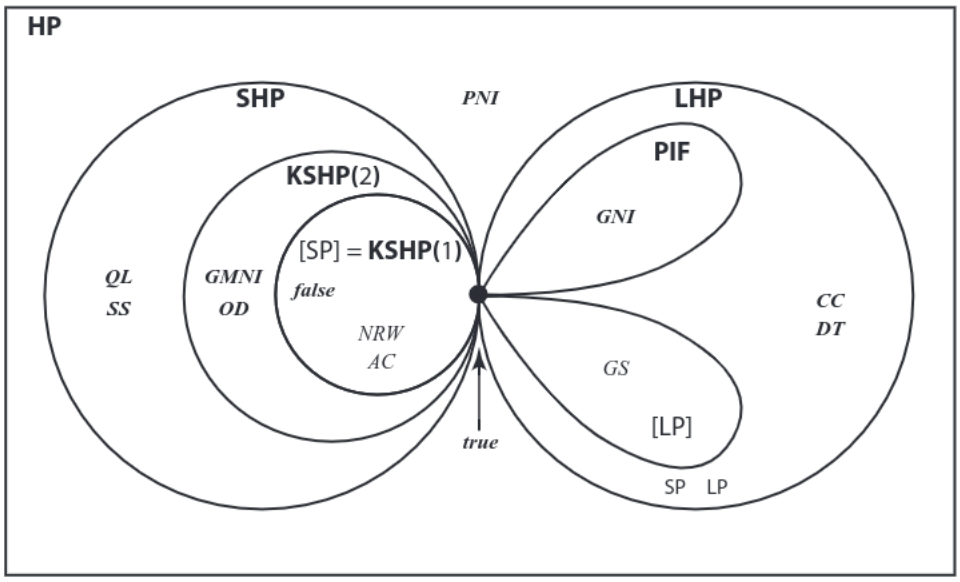

Figure 1. Classification of security policies

The language of confidentiality, integrity, and availability therefore would seem to be orthogonal to hypersafety and hyperliveness. The language of hypersafety and hyperliveness has the advantages of being formalized and providing an orthogonal basis for constructing security policies. In contrast, there is no formalization that simultaneously characterizes confidentiality, integrity, and availability,[14] nor are confidentiality, integrity, and availability orthogonal.[15]

In this work, we developed a theory of hyperproperties that parallels the theory of properties. There is a relatively complete verification methodology for properties: Given a property P, construct a safety property S and a liveness property L such that $P = S \cap L$, then use invariance arguments to verify S and well-foundedness arguments to verify L [4, 5]. We have taken steps toward generalizing this methodology to apply to hyperproperties. Theorem 3 shows that every hyperproperty $\boldsymbol{P}$ can be expressed as the intersection of a safety hyperproperty $\boldsymbol{S}$ and a liveness hyperproperty $\boldsymbol{L}$, and the proof of Theorem 3 shows that $\boldsymbol{S}$ and $\boldsymbol{L}$ can be constructed from $\boldsymbol{P}$. If $\boldsymbol{S}$ is a k-safety hyperproperty, then by Theorem 2, it can be verified using reasoning about safety. It remains an open question whether general methods exist that are relatively complete for verification of safety hyperproperties that are not k-safety, or for liveness hyperproperties.[16] Such methods would complete the verification methodology for hyperproperties. Then, security might take its place as "just another" functional requirement to be verified.

Acknowledgments

We thank Graeme Bailey, Stephen Chong, Dexter Kozen, Ueli Maurer, Andrew Myers, and Tom Roeder for discussions about this work. We also thank Martín Abadi, Borzoo Bonakdarpour, Stephen Chong, Michael George, Leslie Lamport, Jed Liu, John McLean, John Mitchell, Greg Morrisett, Tamara Rezk, Tom Roeder, and Tachio Terauchi for their comments on a draft of this paper.

References

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[2] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[3] Ádám Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In *Proc. of Workshop on Issues in the Theory of Security*, Apr. 2003.

[4] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[5] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[6] R. J. Anderson. A security policy model for clinical information systems. In *Proc. of IEEE Symposium on Security and Privacy*, pages 30–43, 1996.

[14] The closest example of which we are aware is Zheng and Myers [48], who formalize a particular noninterference policy for confidentiality, integrity, and availability.

[15] For example, the requirement that a principal be unable to read a value could be interpreted as confidentiality or unavailability of that value.

[16] If, as discussed at the end of Section 2, the full power of second-order logic is necessary to express hyperproperties, then such methods could not exist. Nonetheless, methods for verifying fragments of the logic might suffice for verifying classes of hyperproperties that correspond to security policies.

[7] G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 100–114, June 2004.

[8] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1–2):109–130, 2002.

[9] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[10] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Belief in information flow. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 31–45, June 2005.

[11] M. R. Clarkson and F. B. Schneider. Hyperproperties. Cornell University Computing and Information Science Technical Report, `http://hdl.handle.net/1813/9480`, Apr. 2008.

[12] R. Focardi and R. Gorrieri. Classification of security properties (Part I: Information flow). In *Foundations of Security Analysis and Design 2000*, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396. Springer, 2001.

[13] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. of IEEE Symposium on Security and Privacy*, pages 11–20, Apr. 1982.

[14] J. W. Gray, III. Probabilistic interference. In *Proc. of IEEE Symposium on Security and Privacy*, pages 170–179, 1990.

[15] J. W. Gray, III. Towards a mathematical foundation for information flow security. In *Proc. of IEEE Symposium on Security and Privacy*, pages 21–34, May 1991.

[16] J. W. Gray, III and P. F. Syverson. A logical approach to multilevel security of probabilistic systems. *Distributed Computing*, 11(2):73–90, 1998.

[17] J. Y. Halpern. *Reasoning About Uncertainty*. MIT Press, 2003.

[18] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[19] L. Lamport. "Sometime" is sometimes "not never": On the temporal logic of programs. In *Proc. of ACM Symposium on Principles of Programming Languages*, pages 174–185, Jan. 1980.

[20] L. Lamport. Basic concepts: Logical foundation. In *Distributed Systems: Methods and Tools for Specification, An Advanced Course*, volume 190 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1985.

[21] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[22] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[23] H. Mantel. Possibilistic definitions of security: An assembly kit. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 185–199, July 2000.

[24] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proc. of IEEE Symposium on Security and Privacy*, pages 161–166, Apr. 1987.

[25] D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, June 1990.

[26] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[27] J. McLean, 2008. Personal communication.

[28] E. Michael. Topologies on spaces of subsets. *Transactions of the American Mathematical Society*, 71(1):152–182, July 1951.

[29] J. Millen. Covert channel capacity. In *Proc. of IEEE Symposium on Security and Privacy*, pages 60–66, Apr. 1987.

[30] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[31] K. R. O'Neill, M. R. Clarkson, and S. Chong. Information-flow security for interactive programs. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 190–201, July 2006.

[32] G. Plotkin. Domains. Available from `http://homepages.inf.ed.ac.uk/gdp/publications/Domains.ps`, 1983.

[33] F. Pottier and V. Simonet. Information flow inference for ML. In *Proc. of ACM Symposium on Principles of Programming Languages*, pages 319–330, Jan. 2002.

[34] R. Pucella and F. B. Schneider. Independence from obfuscation: A semantic framework for diversity. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 230–241, July 2006.

[35] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. of IEEE Symposium on Security and Privacy*, pages 114–127, May 1995.

[36] J. Rushby. Security requirements specifications: How and what? (extended abstract). In *Symposium on Requirements Engineering for Information Security*, Mar. 2001.

[37] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, Jan. 2003.

[38] F. B. Schneider. *On Concurrent Programming*. Springer, 1997.

[39] M. B. Smyth. Power domains and predicate transformers: A topological view. In *Proc. of International Colloquium on Automata, Languages, and Programming*, pages 662–675, July 1983.

[40] M. B. Smyth. Topology. In *Background: Mathematical Structures*, volume 1 of *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[41] D. Sutherland. A model of information. In *National Security Conference*, pages 175–183, 1986.

[42] T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *Proc. of Static Analysis Symposium*, pages 352–367, Sept. 2005.

[43] J. van Benthem and K. Doets. Higher-order logic. In *Elements of Classical Logic*, volume 1 of *Handbook of Philosophical Logic*. D. Reidel Publishing, 1983.

[44] L. Vietoris. Bereiche zweiter Ordnung. *Monatschefte für Mathematik und Physik*, 33:49–62, 1923.

[45] D. Volpano. Safety versus secrecy. In *Proc. of Static Analysis Symposium*, pages 303–311, 1999.

[46] G. Winskel and M. Nielsen. Models for concurrency. In *Semantic Modelling*, volume 4 of *Handbook of Logic in Computer Science*. Oxford University Press, 1994.

[47] S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 29–43, June 2003.

[48] L. Zheng and A. C. Myers. End-to-end availability policies and noninterference. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 272–286, June 2005.

A Summary of Notation

P is a property, $\boldsymbol{H}$ is a hyperproperty, M is a named set of properties, and N is a named set of hyperproperties. We use **bold** to denote "hyper" and sans serif to denote named sets. Predicates and functions always begin with lower case, whereas properties always begin with upper case.

Σ	set of all states
Ψ_{fin}	set of all finite traces
Ψ_{inf}	set of all infinite traces
Ψ	set of all traces
$t[i]$	trace index
$t[..i]$	trace prefix
$t[i..]$	trace suffix
Prop	set of all properties
$\mathcal{P}$	powerset operator
$\mathbb{N}$	the natural numbers
NRW	property "no read then write"
AC	property "access control"
GS	property "guaranteed service"
HP	set of all hyperproperties
$[P]$	lift of property P to equivalent hyperproperty
GMNI	hyperproperty "Goguen and Meseguer's noninterference"
GNI	hyperproperty "generalized noninterference"
OD	hyperproperty "observational determinism"
RT	hyperproperty "average response time"
RC	set of all refinement-closed hyperproperties
SP	set of all safety properties
$\mathcal{P}^{fin}$	finite powerset operator (set of all finite subsets)
Obs	set of all observations
$\leq$	trace (or trace set) prefix
SHP	set of all safety hyperproperties
Ψ_R	set of all relational traces
RHP	set of all relational hyperproperties
RNI	hyperproperty "relational noninterference"
$\mathsf{KSHP}(k)$	set of all k-safety hyperproperties
Sys	set of all systems
SS	hyperproperty "secret sharing"
LP	set of all liveness properties
LHP	set of all liveness hyperproperties
true	maximal hyperproperty
false	minimal hyperproperty
PIF	set of all possibilistic information-flow hyperproperties
DT	hyperproperty "sometimes terminates"
PNI	hyperproperty "probabilistic noninterference"
QL	hyperproperty "quantitative leakage"
CC	hyperproperty "channel capacity"
$\mathcal{O}$	open sets of Plotkin topology
$\uparrow$	completion of observation
$\boldsymbol{\mathcal{O}}$	open sets of our topology
$\boldsymbol{\mathcal{C}}$	closed sets of our topology
$\boldsymbol{\mathcal{D}}$	dense sets of our topology
$\mathfrak{V}_L$	lower Vietoris construction
Ξ	closure under infinite intersection and finite union

B Example Hyperproperties

Bisimulation. Boudol and Castellani [8] give a bisimulation-based noninterference policy for concurrent programs. They model execution as a binary relation $\rightarrow$ on program terms and memories—a program term P and a memory μ step to a new program term P' and memory μ'. Define Σ_P, the set of states for program P, to be the set of pairs of a program term and a memory, $prog(s)$ to be the program term in state s, and $mem(s)$ to be the memory in state s. Define $traces(P)$ to be the set of all traces t such that $prog(t[0])$ is P, and for all i, $t[i] \rightarrow t[i+1]$; this yields a semantic model of P as a set of traces.

Let $=_L$ be an equivalence relation on memories such that $\mu_1 =_L \mu_2$ means μ_1 and μ_2 are indistinguishable to a low observer. Say that state s can *take a step* to state s' when there exists some trace t in $traces(P)$ such that, for some i, we have $t[i] = s$ and $t[i+1] = s'$. Informally, define $\approx_L^P$ (read "bisimilar") to be a binary relation on Σ_P such that if s_1 is bisimilar to s_2, then s_1 and s_2 must have indistinguishable memories to a low observer. Further, if s_1 can take a step to reach state s_1', then either s_1' remains bisimilar to s_2, or s_2 can take a step to reach s_2' where s_1' and s_2' are bisimilar. Formally, bisimilarity $\approx_L^P$ is the largest symmetric binary relation on Σ_P such that

$$\begin{aligned} s_1 \approx_L^P s_2 \implies & mem(s_1) =_L mem(s_2) \\ & \wedge\ (\exists\, t \in traces(P), i \in \mathbb{N}, s_1' \in \Sigma : \\ & t[i] = s_1 \ \wedge\ t[i+1] = s_1' \\ & \implies s_1' \approx_L^P s_2 \\ & \vee\ (\exists\, t' \in traces(P), j \in \mathbb{N}, s_2' \in \Sigma : \\ & t'[j] = s_2 \ \wedge\ t'[j+1] = s_2' \ \wedge \\ & s_1' \approx_L^P s_2')). \end{aligned}$$

Boudol and Castellani define program P to be secure, which we denote as $BCNI(P)$, iff it is bisimilar to itself in all initially low-equivalent memories:

$$\begin{aligned} BCNI(P) \triangleq\ & (\forall\, \mu_1, \mu_2 : \mu_1 =_L \mu_2 \\ & \implies (P, \mu_1) \approx_L^P (P, \mu_2)). \end{aligned}$$

The hyperproperty ***BCNI*** containing all secure programs according to Boudol and Castellani's definition is:

$$\textbf{\textit{BCNI}} \triangleq \{T \in \mathsf{Prop} \mid (\exists\, P : T = \mathit{traces}(P) \wedge \mathit{BCNI}(P))\}. \quad \text{(B.1)}$$

We have shown how to model a particular bisimulation-based definition of noninterference as a hyperproperty. But the example also suggests a general methodology for modeling other bisimulation-based definitions:

1. Model the system used in such a definition as a set of traces, enriching set Σ of states as appropriate.
2. State the bisimulation $\approx$ over Σ.
3. Define the hyperproperty, using $\approx$.

Although the second and third steps in this methodology depend on the particular bisimulation-based definition being modeled, there is a general construction for the first step.

The system model generally used for bisimulation-based definitions is a *labeled transition system*, a triple $(S, L, \rightarrow)$ where S is a set of LTS-states,[17] L is a set of labels, and $\rightarrow$ is a relation on $S \times L \times S$ [30]. Elements of relation $\rightarrow$ are usually notated $s_1 \xrightarrow{\ell} s_2$, meaning that the system has a transition labeled ℓ from LTS-state s_1 to LTS-state s_2.

To model labeled transition system $(S, L, \rightarrow)$ as a set of traces, it suffices to define the state space Σ for systems to be $S \times L$. Given state $s \in \Sigma$, let $st(s)$ denote the LTS-state from s and $lab(s)$ denote the label from s. Define $\mathit{traces}(S, L, \rightarrow)$ to be

$$\{t \mid (\forall\, i \in \mathbb{N} : st(t[i]) \xrightarrow{lab(t[i+1])} st(t[i+1]))\}.$$

Note that this construction would not work with an impoverished notion of state, as observed by Focardi and Gorrieri [12] for states that are elements of L. Defining a state as an element of $S \times L$ captures enough information in the set of traces to express bisimulation-based policies.

Probabilistic noninterference. To formulate probabilistic noninterference as a hyperproperty, we need some notation:

- Let the *low equivalence class* of a finite trace t be denoted $[t]_L$, where

  $$[t]_L \triangleq \{t' \in \Psi_{\mathsf{fin}} \mid \mathit{ev}(t, L) = \mathit{ev}(t', L)\}.$$

  $\mathsf{Pr}_{s,S}([t]_L)$ is the probability that system S, starting in state s, produces an execution low-equivalent to t.

- Let the set of initial states of property T be denoted $\mathit{Init}(T)$, where

  $$\mathit{Init}(T) \triangleq \{s \mid \{s\} \leq T\}.$$

Probabilistic noninterference can then be expressed as

$$\begin{aligned}\textbf{\textit{PNI}} \triangleq \{T \in \mathsf{Prop} \mid (\forall\, s_1, s_2 \in \mathit{Init}(T) :\\ \mathit{ev}(s_1, L) = \mathit{ev}(s_2, L) \implies\\ (\forall\, t \in \Psi_{\mathsf{inf}} :\\ \mathsf{Pr}_{s_1,T}([t]_L) = \mathsf{Pr}_{s_2,T}([t]_L)))\}.\end{aligned}$$

Hyperproperty ***PNI*** is neither hypersafety nor hyperliveness. It is not hypersafety, because even if T fails to be in ***PNI*** because of some equivalence class $[t]_L$, it may be possible to extend T to be in ***PNI*** by extending some prefix of $[t]_L$ in T.[18] Neither is ***PNI*** hyperliveness: A system that deterministically produces two non-low-equivalent traces from two initial low-equivalent states cannot be extended to satisfy ***PNI***.

Quantitative flow. In the model of Clarkson et al. [10], a state has a high component and a low component. A *repeated experiment* on program S is a series of executions of S. In each execution, the initial state must have the same high component but may have a different low component.

We use traces to represent repeated experiments. The first event in the trace is the high component of the initial state. After this follows a series of pairs of low input and low output events. Each low output must have the correct probability of occurring according to S, the initial high input component, and the most recent low input component. The probabilistic behavior of S is modeled by a semantics $[\![S]\!]$ that maps inputs states to output distributions. Thus, $([\![S]\!]s)(s')$ is the probability that S begun in state s terminates in state s'.

Let $\mathit{Syst}(S)$ denote the system of such traces resulting from program S:

$$\begin{aligned}\mathit{Syst}(S) \triangleq \{t \in \Psi_{\mathsf{fin}} \mid (\forall \text{ even } i :\\ \mathit{ev}(t[i], H) = \mathit{ev}(t[0], H)\\ \wedge\ p(t[i+1]) = ([\![S]\!]t[i])(t[i+1]))\}.\end{aligned}$$

Note that $p(s)$ is not defined at all states in these traces. Further, the set of program states must be finite for the probability distributions to be well-defined.

[17] We use the term *LTS-state* to distinguish these from the states defined in Section 2.

[18] Consider two low-equivalent initial states s_1 and s_2 of T. Suppose that the probability of $[t]_L$ from s_1 is 0, but that the probability of some prefix of $[t]_L$ is 1. Further suppose that the probability of $[t]_L$ from s_2 is 1. These assumptions imply that $T \notin$ ***PNI***. But it may be possible to extend the prefix of $[t]_L$ from s_1 such that the trace is now low-equivalent to t. This extended system would now satisfy ***PNI***. Thus ***PNI*** is not hypersafety.

We now formalize the quantity of flow over a trace t. Each pair of states $t[i]$ and $t[i+1]$ can be used to define an *experiment*, which describes how an attacker's beliefs change as a result of observing execution of the program. The quantity of flow in an experiment follows from definitions given in [10], and the quantity of flow over the trace is the sum of the flow for each experiment in the trace:

$$
\begin{aligned}
\mathcal{Q}(b_H, t) &\triangleq \sum_{i=0}^{(|t|-1)/2} \mathcal{Q}(\mathcal{E}(t, i, b_H)) \\
\mathcal{E}(t, i, b_H) &\triangleq \langle pre = \mathcal{E}(t, i-1).post; \\
&\qquad h = ev(t[2i], H); l = ev(t[2i], L); \\
&\qquad post = \delta(b_H, l) | ev(t[2i+1], L) \restriction H \rangle \\
\delta(b_H, l) &\triangleq \lambda s \,.\, b_H(ev(s, H) \cdot \mathsf{Pr}([(ev(s, H) \cup l), \\
&\qquad ev(s, L)]).
\end{aligned}
$$

Hyperproperty ***QL*** is the set of all systems that exhibit less than k bits of flow over any experiment:

$$
\begin{aligned}
\boldsymbol{QL} \triangleq \{T \in \mathsf{Prop} \mid (\exists\, S : T = Syst(S) \\
\wedge\ (\forall\, t \in T, b_H : \mathcal{Q}(b_H, t) \leq k))\}.
\end{aligned}
$$