A Cut Principle for Information Flow

Joshua D. Guttman and Paul D. Rowe
The MITRE Corporation

Abstract—We view a distributed system as a graph of active locations with unidirectional channels between them, through which they pass messages. In this context, the graph structure of a system constrains the propagation of information through it.

Suppose a set of channels is a cut set between an information source and a potential sink. We prove that, if there is no disclosure from the source to the cut set, then there can be no disclosure to the sink. We introduce a new formalization of partial disclosure, called blur operators, and show that the same cut property is preserved for disclosure to within a blur operator. A related compositional principle ensures limited disclosure for a class of systems that differ only beyond the cut.

I. INTRODUCTION

In this paper, we consider information flow in a true-concurrency, distributed model. Events in an execution may be only partially ordered, and locations communicate via synchronous message-passing. Each message traverses a channel. The locations and channels form a directed graph.

Evidently, the structure of this graph constrains the flow of information. Distant locations may have considerable information about each other’s actions, but only if the information in intermediate regions accounts for this. If a kind of information does not traverse the boundary of some portion of the graph (a cut set), then it can never be available beyond that. We represent these limits on disclosure, i.e. kinds of information that do not escape, using blur operators. A blur operator returns a set of behaviors local to the information source; these should be indistinguishable to the observer. Blur operators formalize the semantic content of limited disclosures, and they cover similar ground to other forms of what-dimension declassification [51], [52]. Their definition, however, identifies the principles that localize information flow.

When disclosure from a source to a cut set is limited to within a blur operator, then disclosure to a more distant region is limited to within the same blur operator (see Thm. 28, the cut-blur principle). The cut-blur principle combines our what-dimension declassification with a where-dimension perspective. It gives a criterion that localizes those disclosure limits within a system architecture.

A related result, Thm. 32, supports compositional security. Consider any other system that differs from a given one only in its structure beyond the cut. That system will preserve the flow limitations of the first, assuming that it has the same local behaviors as the first in the cut set. We illustrate this (Examples 33–34) to show that secrecy and anonymity properties of a firewall and a voting system are preserved under some environmental changes. Flow properties of a simple system remain true for more complex systems, if the latter do not distort behavior at the edge of the simple system.

Our model covers many types of systems, including networks, software architectures, virtualized systems, and distributed protocols such as voting systems. Network examples, which involve little local state, are easy to describe, and rely heavily on the directed graph structure. Blur operators highlight their security goals as information-flow properties. Voting systems offer an interesting notion of limited disclosure, since they must disclose the result but not the choices of the individual voters. Their granularity encourages composition, since votes are aggregated from multiple precincts.

Motivation. A treatment of information flow that relies on the graph structure of distributed systems facilitates compositional security design and analysis.

Many systems have a natural graph structure, which is determined early in the design process. Some are distributed systems where the components are on separate platforms, and the communication patterns are a key part of their security architectures. In other cases, the components may be software, such as processes or virtual machines, and the security architecture is largely concerned with their communication patterns. The designers may want to validate that these communication patterns support the information flow goals of the design early in the life cycle. Thm. 32 justifies the designers in concluding that a set of eventual systems all satisfy these security goals, when those systems all agree on “the part that really matters.”

Contributions of this paper. Our main result is the cut-blur principle, Thm. 28, which Thm. 32 brings to compositional form. The definition of blur operator is a supplementary contribution. We show that any reasonable notion of partial disclosure satisfies the conditions for a blur (Lemma 22). We regard these simple structural conditions as giving the “logical form” of composable limited disclosure. The conditions lead to very clean proofs of Thms. 28, 32.

Structure of this paper. After discussing motivating examples (Section II) and some related work (Section III), we introduce our systems, called frames, and their execution model in Section IV. In this static model, the channels connecting different locations do not change during execution. Section V proves the cut-blur principle for the simple case of no disclosure of information at all across the boundary.

Section VI formalizes partial disclosure via blur operators, and Section VII extends the cut idea to blurs (Thms. 28, 32).

Section VIII provides rigorous results to relate our model to the literature. We end by indicating some future directions. Appendix A contains longer proofs, and additional lemmas.
II. TWO MOTIVATING EXAMPLES

We first propose two problems we view in terms of information flow. One is about network filtering; the other concerns anonymity in voting. In each, we want to prove an information flow result once, and then reuse it compositionally under variations that do not affect the core mechanism itself.

Example 1 (Network filtering). Fig. 1 shows a two-router firewall separating the public internet (node i) from two internal network regions n1, n2. The firewall should ensure that any packet originating in the internal regions n1, n2 reaches i only if it satisfies some property of its source and destination addresses, protocol, and port (etc.); we will call these packets exportable. Likewise, any packet originating in i reaches n1, n2 only if it satisfies a related property of its source and destination addresses, protocol, and port (etc.); we will call these packets importable.

These are information flow properties. The policy provides confidentiality for non-exportable packets within n1, n2, ensuring that they are not observable at i. It provides a kind of integrity protection for n1, n2 from non-importable packets from i, ensuring that n1, n2 cannot be damaged, or affected at all, if they are malicious.

We assume here that packets are generated independently, so that (e.g.) no process on a host in n1, n2 generates exportable packets encoding confidential non-exportable packets it has sent or received. If some process on a host is observing packets and coding their contents into packets to a different destination, this is a problem firewalls were not designed to solve, and security administrators worry about it separately.

A firewall configuration enforcing a flow goal against the internet viewed as a single node i should still succeed if i has internal structure. Similarly, the internal regions n1, n2 may vary without risk of security failure.

We will return to this example several times to illustrate how we formalize the system and specify its flow goals. Example 33 proves that some information flow goals of Fig. 1 remain true as the structure of i, n1, n2 varies.

Example 2. As another key challenge, consider an electronic voting system such as ThreeBallot [42]. Fig. 2 shows the voters v1, . . . , vk of a single precinct, their ballot box BB1, a channel delivering the results to the election commission EC, and then a public bulletin board Pub that reports the results.

The ballot box should provide voter anonymity: neither EC nor anyone observing the results Pub should be able to associate any particular vote with any particular voter vi. This also is an information flow goal.

However, elections generally concern many precincts. Fig. 3 contains i precincts, all connected to the election commission EC. Intuitively, a voter vi cannot lose their anonymity in the larger system: BB1 has already anonymized the votes in this first precinct. Accumulating precinct summaries at EC cannot change the causal consequences of BB1’s actions.

We formalize the flow goals of this example in Example 26, and justify Fig. 3 in Example 34.

These simple examples illustrate the payoff from a compositional approach to flow goals. Conclusions about a firewall should be insensitive to changes in the structure of the networks to which it is attached. An anonymity property achieved by a ballot mechanism should be preserved as we collect votes from many precincts. These are situations where we want to design, justify, and then reuse mechanisms, with a criterion ensuring the mechanisms remain safe under changes outside them. Thm. 32 below is the criterion we propose.

III. SOME RELATED WORK

Noninterference and nondeducibility. There is a massive literature on information-flow security; Goguen and Meseguer were key early contributors [20]. Sutherland introduced the non-deducibility idea [53] as a way to formalize lack of information flow, which we have adopted in our “non-disclosure” (Def. 11). Subsequent work has explored a wide range of formalisms, including state machines [47]; process algebras such as CSP [44], [43], [48] and CCS [18], [19], [6]; and bespoke formalisms [36], [29].

Irvine, Smith, and Volpano reinvigorated a language-based approach [25], inherited from Denning and Denning [16], in which systems are programs. Typing ensures that their behaviors satisfy information-flow goals; cf. [50]. Distributed
Declassification. Declassification is a major concern for us. A blur operator (Def. 21) determines an upper bound on what a system may declassify. It may declassify information unless its blur operators require those details to be blurred out. Like escape-hatches [51] or relaxed noninterference [28], this is disclosure along the what-dimension, in the Sabelfeld-Sands classification [52]. The cut-blur principle connects this what declassification to where the processing responsible for the declassification will occur in a system architecture. In this regard, it combines a semantic view of what information is declassified with an architectural view related to intransitive noninterference [47], [54]. Balliu et al. [4] connect what, where, and when declassification via epistemic logic, although without a compositional method.

McCamant and Ernst [34] study quantitative information flow when programs run. A directed acyclic graph representing information flow is generated dynamically from a particular execution or set of executions. The max-flow/min-cut theorem bounds flow in those runs by what can traverse minimal cuts. Apparently, other possible executions may not respect the bounds. Their flow conclusions are not compositional.

Composability and refinement. McCullough first raised the questions of non-determinism and composability of information-flow properties [35], [36]. This was a major focus of work through much of the period since, persisting until today [27], [57], [30], [31], [46], [41]. Mantel, Sands, and Sudbrock [32] use a rely/guarantee method for compositional reasoning about flow in the context of imperative programs. Roscoe [45], [44] offers a definition based on determinism, which is intrinsically composable. Morgan’s [39] programming language treatment clarifies the refinements that preserve security. Our results do not run afoul of the refinement paradox either [26], [39]: our theorems identify the assumptions that ensure that blurs are preserved.

Van der Meyden [55] provides an architectural treatment designed to achieve preservation under refinement. Our work is distinguished from it in offering a new notion of composition, illustrated in Examples 1–2; in focusing on declassification; and in applying uniformly to a range of declassification policies, defined by the blur operators.

Van der Meyden’s work with Chong [10], [11] is most closely related to ours. They consider “architectures,” i.e. directed graphs that express an intransitive noninterference style of what-dimension flow policy. The nodes of an architecture are security domains, intended to represent levels of information sensitivity. The authors define when a (monolithic) deterministic state machine, whose transitions are annotated by domains, complies with an architecture. The main result in [10] is a cut-like epistemic property on the architecture graph: Roughly, any knowledge acquired by a recipient about a source implies that the same knowledge is available at every cut set in the architecture graph.

A primary contrast between this paper and [10] is our distributed execution model. We consider it a more localized link to development, since components are likely to be designed, implemented, and upgraded piecemeal. Chong and van der Meyden focus instead on the specifications, in which sensitivity levels of information (rather than active system components) form the directed graph. This new and unfamiliar specification is needed before analysis. Their epistemic logic allows nested occurrences of the knowledge modality $K_G$, or occurrences of $K_G$ in the hypothesis of an implication. However, this surplus expressiveness is not used in their examples, which do not have nested $K_G$ operators, or occurrences of $K_G$ in the hypothesis of an implication. Indeed, our clean proof methods suggest that our model may have the right degree of generality, and be easy to understand, apply, and enrich.

Recently [11], they label the arrows by functions $f$, where $f$ filters information from its source, bounding visibility to its target. They have not re-established their cut-like epistemic property in the richer model, however. Van der Meyden and Chong’s refinement method [55], [11] applies when the refined system has a homomorphism onto the less refined one. It covers Example 1 but not Example 2, where the refined system contains genuinely new components and events.

We return to related work passim, and in Sections VIII–IX.

IV. FRAMES AND EXECUTIONS

We represent systems by frames. Each frame is a directed graph. Each node, called a location, is equipped with a set of traces defining its possible local behaviors. The arrows are called channels, and allow the synchronous transmission of a message from the location at the arrow tail to the location at the arrow head. Each message also carries some data.

A. A Static Model

In this paper, we will be concerned with a static version of the model, in which channel endpoints are never transmitted from one location to another. Section IX mentions a dynamic alternative, in which these endpoints may be delivered over other channels. Each frame uses three disjoint domains:

Locations $\mathcal{L}O$: Each location $\ell \in \mathcal{L}O$ is equipped with a set of traces, traces($\ell$) and other information, further constrained below.

Channels $\mathcal{C}H$: Each channel $c \in $ is equipped with two endpoints, entry$(c)$ and exit$(c)$. It is intended as a one-directional conduit of data values between the endpoints.

Data values $\mathcal{D}$: Data values $v \in \mathcal{D}$ may be delivered through channels.

We will write $\mathcal{E}\mathcal{P}$ for the set of channel endpoints, which we formalize as $\mathcal{E}\mathcal{P} = \{\text{entry}, \text{exit}\} \times \mathcal{C}H$, although we generally write entry$(c)$ and exit$(c)$ to stand for $\langle\text{entry}, c\rangle$ and $\langle\text{exit}, c\rangle$. 
A frame $\mathcal{F}$ supplies sets of endpoints ends($\ell$) and traces($\ell$) for each location $\ell \in \mathcal{LO}$. When entry(c) $\in$ ends($\ell$) we write sender(c) $\equiv \ell$; when exit(c) $\in$ ends($\ell$) we write rcpt(c) $\equiv \ell$. Thus, sender(c) can send messages on $c$, while rcpt(c) can receive them. We write chans($\ell$) for \{c: sender(c) = $\ell$ or rcpt(c) = $\ell$.\}

We say that $\lambda$ is a label for $\ell$ if $\lambda = (c,v)$ where $c \in$ chans($\ell$) and $v \in \mathcal{D}$; and we categorize labels $c, v$ as:

- **Local to $\ell$** if sender(c) = $\ell$ = rcpt(c);
- **A transmission for $\ell$** if sender(c) = $\ell$ $\neq$ rcpt(c);
- **A reception for $\ell$** if sender(c) $\neq$ $\ell$ = rcpt(c).

With this notation we define frames:

**Definition 3.** Given domains $\mathcal{LO}, CH, D$, $\mathcal{F} = (\text{ends, traces})$ is a frame iff, for each $\ell \in \mathcal{LO}$:

1. ends($\ell$) $\subseteq \mathcal{EP}$ is a set of endpoints such that:
   (a) $\langle c, e \rangle \in$ ends($\ell$) and $\langle c, e \rangle \in$ ends($\ell'$) implies $\ell = \ell'$; and
   (b) there is an $\ell$ such that entry(c) $\in$ ends($\ell$) iff there is an $\ell'$ such that exit(c) $\in$ ends($\ell'$);

2. traces($\ell$) is a prefix-closed set, each trace $t \in$ traces($\ell$) being a finite or infinite sequence of labels $\lambda$.

In this definition, we do not require that the local behaviors traces($\ell$) should be determined in any particular way. They could be specified by associating a program to each location, or a term in a process algebra, or a labeled transition system, or a mixture of these for the different locations.

Each $\mathcal{F}$ determines directed and undirected graphs:

**Definition 4.** If $\mathcal{F}$ is a frame, then the graph of $\mathcal{F}$, written $gr(\mathcal{F})$, is the directed graph $(V, E)$ whose vertices $V$ are the locations $\mathcal{LO}$, and such that there is an edge $\langle \ell_1, \ell_2 \rangle \in E$ iff, for some $c \in CH$, sender(c) = $\ell_1$ and rcpt(c) = $\ell_2$.

The undirected graph $ungr(\mathcal{F})$ has those vertices, and an undirected edge $\langle \ell_1, \ell_2 \rangle$ whenever either $\langle \ell_1, \ell_2 \rangle$ or $\langle \ell_2, \ell_1 \rangle$ is in the edges of $gr(\mathcal{F})$.

**B. Execution semantics**

The execution model for frames uses partially ordered sets of events. The key property is that the events at any single location $\ell$ should be in traces($\ell$). Our semantics is reminiscent of Mattern [33], although his model lacks the underlying graph structure. We require executions to be well-founded, but no later results in this paper depend on that.

**Definition 5 (Events; Executions).** Let $\mathcal{F}$ be a frame, and let $\mathcal{E}$ be a structure $(E, \text{chan, msg})$. The members of $E$ are events, equipped with the functions:

- **chan:** $E \rightarrow CH$ returns the channel of each event; and
- **msg:** $E \rightarrow D$ returns the message passed in each event.

$B = (B, \preceq)$ is a system of events, written $B \in \text{ES}(\mathcal{E})$, iff (i) $B \subseteq E$; (ii) $\preceq$ is a partial ordering on $B$; and (iii) for every $e_1 \in B$, $\{e_0 \in B: e_0 \preceq e_1\}$ is finite.

Hence, $B$ is well-founded. If $B = (B, \preceq)$, we refer to $B$ as ev($B$) and to $\preceq$ as $\preceq_B$.

Now let $B = (B, \preceq) \in \text{ES}(\mathcal{E})$, and define $\text{proj}(B, \ell) = \{e \in B: \text{sender(chan(e))} = \ell \text{ or rcpt(chan(e))} = \ell\}$.

$B$ is an execution, written $B \in \text{Exc}(\mathcal{F})$ iff, for every $\ell \in \mathcal{LO}$,

1. proj($B, \ell$) is linearly ordered by $\preceq$, hence—by the finiteness condition (iii)—a sequence, and
2. proj($B, \ell$) $\in$ traces($\ell$).

We often write $A, A'$, etc., when $A, A' \in \text{Exc}(\mathcal{F})$. The choice between two structures $\mathcal{E}_1, \mathcal{E}_2$ makes little difference: If $\mathcal{E}_1, \mathcal{E}_2$ have the same cardinality, then to within isomorphism they lead to the same systems of events and hence also executions. Thus, we suppress the parameter $\mathcal{E}$, henceforth.

This semantics associates a set of executions with each frame, without imposing any notion of inputs and outputs, or regarding a frame as a program-like function.

**Definition 6.** Let $B_1 = (B_1, \preceq_1), B_2 = (B_2, \preceq_2) \in \text{ES}(\mathcal{E})$.

1. $B_1$ is a substructure of $B_2$ iff $B_1 \subseteq B_2$ and $\preceq_1 \subseteq (\preceq_2 \cap B_1 \times B_1)$.

2. $B_1$ is an initial substructure of $B_2$ iff $B_1$ is a substructure of $B_2$, and for all $y \in B_1$, if $x \preceq_2 y$, then $x \in B_1$.

**Lemma 7.**

1. If $B_1$ is a substructure of $B_2 \in \text{ES}(\mathcal{F})$, then $B_1 \in \text{ES}(\mathcal{F})$.

2. If $B_1$ is an initial substructure of $B_2 \in \text{Exc}(\mathcal{F})$, then $B_1 \in \text{Exc}(\mathcal{F})$.

3. Being an execution is preserved under chains of initial substructures: Suppose that $\{B_i\}_{i \in \mathbb{N}}$ is a sequence where each $B_i \in \text{Exc}(\mathcal{F})$, such that $i \leq j$ implies $B_i$ is an initial substructure of $B_j$. Then $\bigcup_{i \in \mathbb{N}} B_i \in \text{Exc}(\mathcal{F})$.///

**Example 8 (Network with filtering).** To localize our descriptions of functionality, we expand the network of Fig 1; see Fig. 4. Regions are displayed as $\bullet$ routers, as $\square$; and interfaces, as $\triangle$. When a router has an interface onto a segment, a pair of locations—representing that interface as used in each direction—lie between this router and each peer router [21].

Let Dir = \{inb, outb\} represent the inbound direction and the outbound directions from routers, respectively. Suppose Rt is a set of routers $r$, each with a set of interfaces intf($r$), and a set of network regions $Rg$ containing end hosts.

Each member of $\text{Rt, Rg}$ is a location. Each interface-direction pair \langle i, r \rangle = (\bigcup_{i \in \text{Rt}} \text{intf}(r)) \times \text{Dir}$ is also a location. The channels are those shown. Each interface has a pair of channels that allow datagrams to pass between the router and the interface, and between the interface and an adjacent entity. We also include a self-loop channel at each network region $i, n_1, n_2$; it represents transmissions and receptions among the hosts and network infrastructure coalesced into the region.

Thus:

- $\mathcal{LO} = \text{Rt} \cup \text{Rg} \cup (\bigcup_{i \in \text{Rt}} \text{intf}(r)) \times \text{Dir}$;

- $\mathcal{CH} = \{(\ell_1, \ell_2) \in \mathcal{LO} \times \mathcal{LO}: \ell_1 \text{ delivers datagrams directly to } \ell_2\};$

- $\mathcal{D}$ = the set of IP datagrams;

- ends($\ell$) = \{entry($\ell, \ell_2$): ($\ell, \ell_2$) $\in$ CH\} $\cup$ \{exit($\ell_1, \ell$): ($\ell_1, \ell$) $\in$ CH\}, for each $\ell \in \mathcal{LO}$.
The traces are easily specified. Each router \( r \in R_t \) receives packets from inbound interfaces, and chooses an outbound interface for each. Its state is a set of received but not yet routed datagrams, and the sole initial state is \( \emptyset \). The transition relation, when receiving a datagram, adds it to this set. When transmitting a datagram \( d \) in the current set, it removes \( d \) from the next state and selects an outbound channel as determined by the routing table. For simplicity, the routing table is an unchanging part of determining the transition relation.

A directed interface enforces filtering rules. The state again consists of the set of received but not-yet-processed datagrams. The transition relation uses an unchanging filter function to determine, for each datagram, whether to discard it or retransmit it.

If \( n \in R_g \) is a region, its state is the set of datagrams it has received and not yet retransmitted. It can receive a datagram; transmit one from its state; or else initiate a new datagram. If it is assumed to be well-configured, these all have source address in a given range of IP addresses. Otherwise, the source addresses may be arbitrary.

If the router is executing other sorts of processing, for instance Network Address Translation or the IP Security Protocols, then the behavior is slightly more complex \([1], [21]\), but sharply localized. Many other problems can be viewed as frames. Beyond voting schemes (Ex. 2), attestation architectures \([13]\) and other secure virtualized systems are, at one level, sets of virtual machines communicating through one-directional channels.

**Partially vs. totally ordered executions.** Def. 5 does not require the ordering \( \preceq \) of “occurring before” to be total. When events occur on different channels, neither has to precede the other. Thus, our executions need not be sequential.

This has three advantages. First, it is more inclusive, since executions with total orders satisfy our definition as do those with (properly) partial orders. Indeed, the main claims of this paper remain true when restricted to executions that are totally ordered. Second, reasoning is simplified. We do not need to interleave events when combining two local executions to construct a global one, as encapsulated in the proofs of Lemmas 15, 41. Nor do we need to “compact” events, when splitting off a local execution, as we would if we used a particular index set for sequences. This was probably an advantage to us in developing these results. Third, the minimal partial order is a reflection of causality, which can be used also to reason about independence. We expect this to be useful in future work.

There is also a disadvantage: unfamiliarity. It requires some caution. Moreover, mechanized theorem provers have much better support for induction over sequences than over well-founded orders. This inconvenienced a colleague who used PVS \([40]\) to formalize parts of this work, and eventually chose to use totally ordered executions for induction-oriented proofs. With that difference, Thms. 28 and 32 have been confirmed in PVS, as have the basic properties of Example 34.

### V. NON-DISCLOSURE

Following Sutherland \([53]\), we think of information flow in terms of *deducibility* or *disclosure*. A participant observes part of the system behavior, trying to draw conclusions about a different part. If his observations exclude some possible behaviors of that part, then he can deduce that those behaviors did not occur. His observations have disclosed something.

These observations occur on a set of channels \( C_o \subseteq C_h \), and the deductions constrain the events on a set of channels \( C_s \subseteq C_h \). \( C_o \) is the set of observed channels, and \( C_s \) is the set of source channels. The observer has access to the events on the channels in \( C_o \) in an execution, using these events to learn about what happened at the source. The observed events may rule out some behaviors on the channels \( C_s \).

**Definition 9.** Let \( C \subseteq C_h \), and \( B \in ES(\mathcal{F}) \).

1. The restriction \( B \| C \) of \( B \) to \( C \) is \((B_0, R)\), where
   \[
   B_0 = \{ e \in B : \text{chan}(e) \in C \}, \quad R = (\times \cap B_0 \times B_0).
   \]
2. \( \mathcal{F} \) is a \( C \)-run iff for some \( A \in \text{Exc}(\mathcal{F}) \), \( B = A \| C \). We write \( C \)-runs(\( \mathcal{F} \)), or sometimes \( C \)-runs, for the set of \( C \)-runs of \( \mathcal{F} \). A local run is a member of \( C \)-runs for the relevant \( C \).
3. \( J_{C \setminus C'}(B) \) gives the \( C' \)-runs compatible with a \( C \)-run \( B \):
   \[
   J_{C \setminus C'}(B) = \{ A \upharpoonright C' : A \in \text{Exc}(\mathcal{F}) \text{ and } A \upharpoonright C = B \}.
   \]

\( B \| C \in ES(\mathcal{F}) \) by Lemma 7. In \( J_{C \setminus C'}(B) \), the lower right index \( C' \) indicates what type of local run \( B \) is. The lower left index \( C \) indicates the type of local runs in the resulting set.

**Lemma 8.**

1. \( C \)-runs = \( J_{\emptyset \setminus \emptyset}(\emptyset, \emptyset) \), i.e. the local runs at \( C \) are all those compatible with the empty event set \( (\emptyset, \emptyset) \) at the empty set of channels.
2. \( B \notin \mathcal{F} \) implies \( J_{C \setminus C'}(B) = \emptyset \).
3. \( B \in \mathcal{F} \) implies \( J_{C \setminus C'}(B) = \{ B \} \).
4. \( J_{C \setminus C'}(B) \subseteq C' \)-runs.
A witnesses for \( B' \in J_{C' \rightarrow C}(B) \) iff \( A \in \operatorname{Exc}(F) \), \( B = A \mid C \), and \( B' = A \mid C' \).

No disclosure means that any observation \( B \) at \( C \) is compatible with everything that could have occurred at \( C' \), where compatible means that there is some execution that combines the local \( C \)-run with the desired \( C' \)-run.

We summarize “no disclosure” by the Leibnizian slogan: Everything possible is compossible, “compossible” being his coinage meaning possible together. If \( B, B' \) are each separately possible—being \( C, C' \)-runs respectively—then there’s an execution \( A \) combining them, and restricting to each of them.

**Definition 11.** \( F \) has no disclosure from \( C \) to \( C' \), iff for every \( C \)-run \( B \), \( J_{C' \rightarrow C}(B) = C' \)-runs.

### A. Symmetry of disclosure

Like Shannon’s mutual information and Sutherland’s non-deducibility [53], “no disclosure” is symmetric:

**Lemma 12.** 1. \( B' \in J_{C' \rightarrow C}(B) \) iff \( B \in J_{C \rightarrow C'}(B') \).

2. \( F \) has no disclosure from \( C \) to \( C' \) iff \( F \) has no disclosure from \( C' \) to \( C \).

**Proof.** 1. By the definition, \( B' \in J_{C' \rightarrow C}(B) \) iff there exists an execution \( B_1 \) such that \( B_1 \mid C = B \) and \( B_1 \mid C' = B' \). Which is equivalent to \( B \in J_{C \rightarrow C'}(B') \).

2. There is no disclosure from \( C' \) to \( C \) iff for every \( C \)-run \( B \) and \( C' \)-run \( B' \), \( B' \in J_{C' \rightarrow C}(B) \). By Clause 1, this is the same as \( B \in J_{C \rightarrow C'}(B') \).

Because of this symmetry, we speak of no disclosure between \( C \) and \( C' \).

**Lemma 13.** 1. Suppose \( C_0 \subseteq C_1 \) and \( C_0' \subseteq C_1' \). If \( F \) has no disclosure from \( C_0 \) to \( C_0' \), then \( F \) has no disclosure from \( C_0' \) to \( C_0 \).

2. When \( C_1, C_2, C_3 \subseteq CH \),

\[
J_{C_1 \rightarrow C_2}(B_1) \subseteq \bigcup_{B_2 \in J_{C_2 \rightarrow C_3}(B_1)} J_{C_3 \rightarrow C_2}(B_2).
\]

This is not always an equality. \( B_1 \in C_1 \)-runs and \( B_2 \in C_3 \)-runs may make incompatible demands on a location \( \ell \). The location \( \ell \) may have endpoints on channels in both \( C_1 \) and \( C_3 \); or paths may connect \( \ell \) to both \( C_1 \) and \( C_3 \) without traversing \( C_2 \).

The partial order semantics means that no arbitrary interleaving is needed to create the instance \( A \). Lemma 15 is in fact a corollary of Lemma 31, which makes an analogous assertion about a pair of overlapping frames.

**Theorem 16.** Let cut be an undirected cut between \( src, obs \) in \( F \). If there is no disclosure between \( src \) and \( cut \), then there is no disclosure between \( src \) and \( obs \).

**Proof.** Suppose that \( B_s \in src \)-runs and \( B_o \in obs \)-runs. We must show \( B_s \in J_{src \rightarrow obs}(B_o) \). To apply Lemma 15, let \( A \in \operatorname{Exc}(F) \) such that \( B_o = A \mid obs \), \( A \) exists by the definition of obs-run. Letting \( B_s = A \mid cut \), we have \( B_s \in J_{src \rightarrow obs}(B_o) \).

Since there is no disclosure between cut and \( src \), \( B_s \in J_{src \rightarrow obs}(B_o) \), and Lemma 15 applies.

**Example 17.** In Fig. 4 let \( r_1 \) be configured to discard all inbound packets, and \( r_2 \) to discard all outbound packets. Then the empty event system is the only member of \( \{c_1, c_2\} \)-runs. Hence there is no disclosure between \( \text{chans}(i) \) and \( \{c_1, c_2\} \). By Thm. 16, there is no disclosure to \( \text{chans}(\{n_1, n_2\}) \).
Disconnected portions of a frame cannot interfere:

Corollary 18. If there is no path between src and obs in ungr\((F)\), then there is no disclosure between them.

Proof. Then cut = ∅ is an undirected cut set, and there is only one cut-run, namely the empty system of events. It is thus compatible with all src-runs.

Thm. 16 and its analogue Thm. 28, while reminiscent of the max flow/min cut principle (cf. e.g. [14, Sec. 26.2]), are however quite distinct from it, as the latter depends essentially on the quantitative structure of network flows. Our results may also seem reminiscent of the Data Processing Inequality, stating that when three random variables \(X, Y, Z\) form a Markov chain, the mutual information \(I(X; Z) \leq I(X; Y)\). Indeed, Thm. 16 entails the special case where \(I(X; Y) = 0\), choosing \(gr(F)\) to be a single path \(X \rightarrow Y \rightarrow Z\). For more on quantitative information flow, see the conclusion (Sec. IX).

VI. BLUR OPERATORS

We will now adapt our theory to apply to partial disclosure as well as no disclosure. An observer learns something about a source of information when his observations are compatible with a proper subset of the behaviors possible for the source. Thus, the natural way to measure what has been learnt is the decrease in the set of possible behaviors at the source (see among many sources of this idea e.g. [17], [3]).

This starting point suggests focusing, for every frame and regions of interest src \(\subseteq C H\) and obs \(\subseteq C H\), on the compatibility equivalence relations on src-runs:

Definition 19. Let src, obs \(\subseteq C H\). If \(B_1, B_2\) are src-runs, we say that they are obs-equivalent, and write \(B_1 \approx_{\text{obs}} B_2\), iff, for all obs-runs \(B_o\), \(B_1 \in J_{\text{src}}\text{-obs}(B_o)\) iff \(B_2 \in J_{\text{src}}\text{-obs}(B_o)\).

Lemma 20. For each obs and \(B_o\) in obs-runs:
1. \(\approx_{\text{obs}}\) is an equivalence relation;
2. \(J_{\text{src}}\text{-obs}(B_o)\) is a union of \(\approx_{\text{obs}}\)-equivalence classes; let \(\{S_i\}_{i \in I}\) be the family of all \(\approx_{\text{obs}}\)-equivalence classes. For some \(I_0 \subseteq I\), \(J_{\text{src}}\text{-obs}(B_o) = \bigcup_{i \in I_0} S_i\).

No disclosure means that all src-runs are obs-equivalent, i.e. \(I_0\) always equals \(I\). Any notion of partial disclosure must respect obs-equivalence, since no observations can possibly “split” apart obs-equivalent src-runs. Partial disclosures always respect unions of obs-equivalence classes.

Rather than working directly with these unions of obs-equivalence classes, we instead focus on functions on sets of runs that satisfy three properties. These properties express the structural principles on partial disclosure that make our cut-blur and compositional principles hold. We call operators satisfying the properties blur operators. Lemma 22 shows that an operator that always returns unions of obs-equivalence classes is necessarily a blur operator.

When we want to prove results about all notions of partial disclosure, we prove them for all blur operators. When we want to show a particular relation is a possible notion of partial disclosure, we show that it generates an equivalence relation; Lemma 22 then justifies us in applying Thms. 28, 32.

Definition 21. A function \(f\) on sets is a blur operator iff it satisfies:

Inclusion: For all sets \(S, S \subseteq f(S)\);

Idempotence: \(f\) is idempotent, i.e. for all sets \(S, f(f(S)) = f(S)\);

Union: \(f\) commutes with unions: If \(S_{a \in I}\) is a family indexed by the set \(I\), then

\[
f(\bigcup_{a \in I} S_a) = \bigcup_{a \in I} f(S_a)\quad(1)
\]

\(S\) is \(f\)-blurred iff \(f\) is a blur operator and \(S = f(S)\).

By Idempotence, \(S\) is \(f\)-blurred iff it is in the range of the blur operator \(f\). Since \(S = \bigcup_{a \in S}\{a\}\), the \(\text{Union}\) property says that \(f\) is determined by its action on the singleton subsets of \(S\). Thus, Inclusion could have said \(a \in f(\{a\})\).

Monotonicity also follows from the \(\text{Union}\) property: if \(S_1 \subseteq S_2\), then \(S_2 = S_0 \cup S_1\), where \(S_0 = S_2 \setminus S_1\). Thus \(f(S_2) = f(S_0) \cup f(S_1)\), so \(f(S_1) \subseteq f(S_2)\).

Lemma 22. Suppose that \(A\) is a set, and \(\mathcal{R}\) is a partition of the elements of \(A\). There is a unique function \(f_{\mathcal{R}}\) on sets \(S \subseteq A\) such that

1. \(f_{\mathcal{R}}(\{a\}) = S\) iff \(a \in S\) and \(S \in \mathcal{R}\);
2. \(f_{\mathcal{R}}\) commutes with unions (Eqn. 1).

Moreover, \(f_{\mathcal{R}}\) is a blur operator.

Proof. Since \(S = \bigcup_{a \in S}\{a\}\), \(f_{\mathcal{R}}(S)\) is uniquely defined by the union principle (Eqn. 1).

Inclusion and \(\text{Union}\) are immediate from the form of the definition. Idempotence holds because being in the same \(\mathcal{R}\)-equivalence class is transitive.

Although every equivalence relation determines a blur operator, the converse is not true: Not every blur operator is of this form. For instance, let \(A = \{a, b\}\), and let \(f(\{a\}) = \{a\}\), \(f(\{b\}) = f(\{a, b\}) = \{a, b\}\). However, by Lemma 20 (cf. [24, Prop. 8]), useful partial disclosure is of this form:

Lemma 23. If \(S = J_{\text{src}}\text{-obs}(B_o)\) is \(f\)-blurred, and \(B_o \in S\), then \(f(\{B_o\})\) is a union of \(\approx_{\text{obs}}\)-equivalence classes.

The importance of Def. 21 is to identify the proof principles that make Thm. 28 true. The intuition comes from blurring an image: The viewer no longer knows the details of the scene, but only that it was some scene which, when blurred, would look like this, as the following example indicates.

Example 24. Imaginary Weather Forecasting Inc. (IWF) sells tailored, high-resolution weather data and forecasts to airlines, airports, etc., and low-resolution weather data more cheaply to TV and radio stations. IWF’s low-tier subscribers should not learn higher resolution data than they have paid for. There is some disclosure about high resolution data because (e.g.) when low-tier subscribers see warm temperatures, they know that
the high-resolution data is inconsistent with snow. We can formalize this partial disclosure as a blur.

Suppose IWF creates its low-resolution data \( d_L \) by applying a lossy compression function \( \text{comp} \) to high-resolution data \( d_H \). When low-tier subscribers receive \( d_L \), they know that the high-resolution data IWF measured from the environment is some element of \( \text{comp}^{-1}(d_L) = \{ d_H : \text{comp}(d_H) = d_L \} \). These sets are \( f \)-blurred where \( f(\{d_H\}) = \{ d_H' : \text{comp}(d_H') = \text{comp}(d_H) \} \).

Curiously, IWF wants the low-tier customer, who receives one set of outputs, not to be able to infer too much about the outputs delivered to the high-tier customers. The inputs to the system—sensor values for temperature, wind, pressure etc. at different locations—are not of high value [22].

We will study information disclosure to within blur operators \( f \), which we interpret as meaning \( J_{C \cap \mathcal{E}}(B_0) \) is \( f \)-blurred. This is an “upper bound” on how much information about the local run at \( C \) may be disclosed when \( B \) is observed. The observer will know an \( f \)-blurred set \( S \in \mathcal{P}(C'-\text{runs}) \) to which the behavior at \( C' \) belongs, without being able to infer anything finer than this \( f \)-blurred set.

**Definition 25.** Let \( \text{obs}, \text{src} \subseteq CH \) and \( f : \mathcal{P}(\text{src-runs}) \rightarrow \mathcal{P}(\text{src-runs}) \).

\( \mathcal{F} \) restricts disclosure from \( \text{src} \) to \( \text{obs} \) to within \( f \) iff \( f \) is a blur operator and \( J_{\text{obs}}(B_0) \) is \( f \)-blurred, for every \( B_0 \in \text{obs-runs} \).

We also say that \( \mathcal{F} \) \( f \)-limits src-to-obs flow.

At one extreme, no-disclosure is disclosure to within a blur operator, namely the one that ignores \( S \) and adds all \( C' \)-runs:

\[
\mathcal{F}_S(S) = \{ A \upharpoonright C' : A \in \text{Exc}(\mathcal{F}) \}.
\]

At the other extreme, the maximally permissive security policy is disclosure to within the identity \( \mathcal{F}_S(S) = S \). The blur \( \mathcal{F}_S \) shows that every frame restricts disclosure to within some blur operator. Every set is a union of \( \mathcal{F}_S \)-blurred sets.

\( \mathcal{F} \) may \( f \)-limit src-to-obs flow even when the intersection \( \text{obs} \cap \text{src} \) is non-empty, as long as \( f \) is not too fine-grained; see below (Def. 38).

**Example 26.** Suppose that \( \mathcal{F} \) is an electronic voting system such as ThreeBallot [42]. Some locations \( \mathcal{L}_{\mathcal{E}} \) are run by the election commission. We will regard the voters themselves as a set of locations \( \mathcal{L}_V \). Each voter delivers a message containing, in some form, his vote for some candidate.

The election officials observe the channels connected to \( \mathcal{L}_{\mathcal{E}} \), i.e. \( \text{chans}(\mathcal{L}_{\mathcal{E}}) \). To determine the correct outcome, they must infer a property of the local run at \( \text{chans}(\mathcal{L}_V) \), namely, how many votes for each candidate occurred. However, they should not find out which voter voted for which candidate [15].

We formalize this via a blur operator. Suppose \( B' \in \text{chans}(\mathcal{L}_V) \)-runs is a possible behavior of all voters in \( \mathcal{L}_V \). Suppose that \( \pi \) is a permutation of \( \mathcal{L}_V \). Let \( \pi \cdot B' \) be the behavior in which each voter \( \ell \in \mathcal{L}_V \) casts not his own actual vote, but the vote actually cast by \( \pi(\ell) \). That is, \( \pi \)

represents one way of reallocating the actual votes among different voters. Now for any \( S \subseteq \text{chans}(\mathcal{L}_V) \)-runs let

\[
f_0(S) = \{ \pi \cdot B' : B' \in S \land \pi \text{is a permutation of } \mathcal{L}_V \}.
\]

This is a blur operator: (i) the identity is a permutation; (ii) permutations are closed under composition; and (iii) Eqn. 2 implies commutation with unions. The election commission should learn nothing about the votes of individuals, meaning that, for any \( B \in \text{chans}(\mathcal{L}_{\mathcal{E}}) \)-runs the commission could observe, \( J_{\text{chans}(\mathcal{L}_V) \otimes \text{chans}(\mathcal{L}_{\mathcal{E}})}(B) \) is \( f_0 \)-blurred. Permutations of compatible voting patterns are also compatible.

This example is easily adapted to other considerations. For instance, the commissioners of elections are also voters, and they know how they voted themselves. Thus, we could define a (narrower) blur operator \( f_1 \) that only uses the permutations that leave commissioners’ votes fixed.

In fact, voters are often divided among different precincts, and tallies are reported on a per-precinct basis. Thus, we have sets \( V_1, \ldots, V_k \) of voters registered at the precincts \( P_1, \ldots, P_k \) respectively. The relevant blur function says that we can permute the votes of any two voters \( v_1, v_2 \in V_i \) within the same precinct. One cannot permute votes between different precincts, since that could change the tallies in the individual precincts.

**Example 27.** Suppose in Fig. 4: The inbound interface from \( i \) to router \( r_1 \) discards downward-flowing packets unless their source is an address in \( i \) and the destination is an address in \( n_1, n_2 \). The inbound interface for downward-flowing to router \( r_2 \) discards packets unless the destination address is the IP for a web server \( \text{www} \) in \( n_1 \), and the destination port is 80 or 443, or else their source port is 80 or 443 and their destination port is \( \geq 1024 \).

We filter outbound (upward-flowing) packets symmetrically.

A packet is importable iff its source address is in \( i \) and either its destination is \( \text{www} \) and its destination port is 80 or 443; or else its destination address is in \( n_1, n_2 \), its source port is 80 or 443, and its destination port is \( \geq 1024 \).

It is exportable iff, symmetrically, its destination address is in \( i \) and either its source is \( \text{www} \) and its source port is 80 or 443; or else its source address is in \( n_1, n_2 \), its destination port is 80 or 443, and its source port is \( \geq 1024 \).

We will write select \( B \cdot p \) for the result of selecting those events \( e \in \text{ev}(B) \) that satisfy the predicate \( p(e) \), restricting \( \leq \) to the selected events. Now consider the operator \( f_1 \) on \( \text{chans}(i) \)-runs generated as in Lemma 22 from the equivalence relation:

\[
B_1 \approx_i B_2 \text{ iff they agree on all importable events, i.e.:}
\]

\[
\text{select } B_1 (\lambda e . \text{msg}(e) \text{ is importable}) \equiv \text{select } B_2 (\lambda e . \text{msg}(e) \text{ is importable}).
\]

The router configurations mentioned above are intended to ensure that there is \( f_i \)-limited flow from \( \text{chans}(i) \) to \( \text{chans}(\{n_1, n_2\}) \). This is an integrity condition; it is meant to ensure that systems in \( n_1, n_2 \) cannot be affected by bad (i.e. non-importable) packets from \( i \).
Outbound, the blur \( f_c \) on chans(\( \{n_1, n_2\} \)) runs is generated from the equivalence relation:
\[ B_1 \approx_c B_2 \text{ iff they agree on all exportable events, i.e.:} \]
- select \( B_1 (\lambda e . \text{msg}(e) \text{ is exportable}) \)
- select \( B_2 (\lambda e . \text{msg}(e) \text{ is exportable}) \).

The router configurations are also intended to ensure that there is \( f_c \)-limited flow from chans(\( \{n_1, n_2\} \)) to chans(\( i \)).

This is a confidentiality condition; it is meant to ensure that external observers learn nothing about the non-exportable traffic, which was not intended to exit the organization.

In this example, transmission of an exportable packet is never dependent on reception of a non-exportable packet, and similarly for importable packets. In applications lacking this simplifying property, proving flow limitations is harder.

### VII. THE CUT-BLUR PRINCIPLE

The symmetry of non-disclosure (Lemma 12) no longer holds for disclosure within a blur. We have, however, the natural extension of Thm 16:

**Theorem 28** (Cut-Blur Principle). Let cut be an undirected cut between src, obs in \( F \). If \( F \) \( f \)-limits src-to-cut flow, then \( F \) \( f \)-limits src-to-obs flow.

**Proof.** By the hypothesis, \( f \) is a blur operator. Let \( B_a \) be a obs-run. We want to show that \( J_{\text{obs}}(B_a) \) is an \( f \)-blurred set, i.e. \( J_{\text{obs}}(B_a) = f(J_{\text{obs}}(B_a)) \).

For convenience, let \( S_c = J_{\text{cut}}(B_a) \).

By Lemma 15, \( J_{\text{obs}}(B_a) = \bigcup_{B_c \in S_c} J_{\text{sc}}(B_c) \). Thus, we must show that the latter is \( f \)-blurred.

By the assumption that each \( J_{\text{sc}}(B_c) \) is \( f \)-blurred and by idempotence, \( J_{\text{sc}}(B_c) = f(J_{\text{sc}}(B_c)) \). Now:
\[
\bigcup_{B_c \in S_c} J_{\text{sc}}(B_c) = \bigcup_{B_c \in S_c} f(J_{\text{sc}}(B_c)) = f\left( \bigcup_{B_c \in S_c} J_{\text{sc}}(B_c) \right),
\]

applying the union property (Eqn. 1). Hence, \( \bigcup_{B_c \in S_c} J_{\text{sc}}(B_c) \) is \( f \)-blurred.

This is why we introduced the Union principle Eqn. 1, rather than simply considering all closure operators [37]. Eqn. 1 distinguishes the closure operators that allow the “long distance reasoning” summarized in the proof.

**Example 29.** The frame of Example 27 has \( f_c \)-limited flow from chans(\( i \)) to the cut \( \{c_1, c_2\} \). Thus, it has \( f_c \)-limited flow from chans(\( \{n_1, n_2\} \)) to chans(\( \{c_1, c_2\} \)).

It also has \( f_c \)-limited flow from chans(\( \{n_1, n_2\} \)) to the cut \( \{c_1, c_2\} \). This implies \( f_c \)-limited flow to chans(\( i \)).

### A. A Compositional Relation between Frames

Our next technical result gives us a way to “transport” a blur security property from one frame \( F_1 \) to another frame \( F_2 \). It assumes that the two frames share a common core, some set of locations \( L_0 \). These locations should hold the same channel endpoints in each of \( F_1, F_2 \), and should engage in the same traces. The boundary separating \( L_0 \) from the remainder of \( F_1, F_2 \) necessarily forms a cut set cut. Assuming that the local runs at cut are respected, blur properties are preserved from \( F_1 \) to \( F_2 \).

**Definition 30.** A set \( L_0 \) of locations is shared between \( F_1 \) and \( F_2 \) iff \( F_1, F_2 \) are frames with locations \( L_0 \), endpoints \( \text{ends}_1, \text{ends}_2 \), and traces \( \text{traces}_1, \text{traces}_2 \), resp., where \( L_0 \subseteq \mathcal{L}_1 \cap \mathcal{L}_2 \), and for all \( \ell \in L_0 \), \( \text{ends}_1(\ell) = \text{ends}_2(\ell) \) and \( \text{traces}_1(\ell) = \text{traces}_2(\ell) \).

When \( L_0 \) is shared between \( F_1 \) and \( F_2 \), let:
- \( \text{left}_0 = \{ c \in \mathcal{C}_1 \mid \text{both endpoints of } c \text{ are locations } \ell \in L_0 \} \),
- \( \text{cut}_0 = \{ c \in \mathcal{C}_1 \mid \text{ exactly one endpoint of } c \text{ is a location } \ell \in L_0 \} \), and
- \( \text{right}_0 = \{ c \in \mathcal{C}_1 \mid \text{ neither endpoint of } c \text{ is a location } \ell \in L_0 \} \), for \( i = 1, 2 \).

We will also use \( C \)-runs and \( C \)-runs to refer to the local runs of \( C \) within \( F_1 \) and \( F_2 \), resp.; and \( J^1_{\text{cut}}(B) \) and \( J^2_{\text{cut}}(B) \) will refer to the compatible \( C \)-runs in the frames \( F_1 \) and \( F_2 \), resp.

Indeed, \( \text{cut}_0 \) is an undirected cut between \( \text{left}_0 \) and \( \text{right}_0 \) in \( F_i \), for \( i = 1 \) and 2. In an undirected path that starts in \( \text{left}_0 \) and never traverses \( \text{cut}_0 \), each arc always has both ends in \( L_0 \). We next prove a two-frame analog of Lemma 15.

**Lemma 31.** Let \( L_0 \) be shared between frames \( F_1, F_2 \). Let \( \text{src} \subseteq \text{left}_0 \) and \( B_c \in \text{cut}_0 \text{-runs}_1 \cap \text{cut}_0 \text{-runs}_2 \).

1. \( J^1_{\text{src}}(B_c) = J^2_{\text{src}}(B_c) \).
2. Assume \( \text{cut}_0 \text{-runs}(F_2) \subseteq \text{cut}_0 \text{-runs}(F_1) \). Let \( \text{obs} \subseteq \text{right}_2 \) and \( B_c \in \text{obs}_0 \text{-runs}_2 \). Then
\[
J^2_{\text{obs}}(B_c) = \bigcup_{B_c \in \text{cut}_0 \text{-obs}_2} J^1_{\text{src}}(B_c).
\]

Part 1 states that causality acts locally. The variable portions \( \text{right}_1, \text{right}_2 \) of \( F_1 \) and \( F_2 \) can affect what happens in their shared part left. But it does so only by changing which cut0-runs are possible. Whenever both frames agree on any \( B_c \in \text{cut}_0 \text{-runs}_1 \cap \text{cut}_0 \text{-runs}_2 \), then the left-run runs compatible with \( B_c \) are the same. Distant effects from right1 to left occur only via local runs at the boundary cut0.

The assumption \( \text{cut}_0 \text{-runs}(F_2) \subseteq \text{cut}_0 \text{-runs}(F_1) \) in Part 2 and Thm. 32 is meant to limit this variability in one direction.

**Theorem 32.** Suppose that \( L_0 \) is shared between frames \( F_1, F_2 \), and assume \( \text{cut}_0 \text{-runs}(F_2) \subseteq \text{cut}_0 \text{-runs}(F_1) \). Consider any \( \text{src} \subseteq \text{left}_0 \) and \( \text{obs} \subseteq \text{right}_2 \). If \( F_1 \) \( f \)-limits src-to-cut flow, then \( F_2 \) \( f \)-limits src-to-obs flow.

The proof is similar to the proof of the cut-blur principle, which effectively results from it by replacing Lemma 31 by
Lemma 15, and omitting the subscripts on frames and their local runs. The cut-blur principle is in fact the corollary of Thm. 32 for \( F_1 = F_2 \).

**B. Two Applications**

Thm. 32 is useful as a compositional principle. It implies that in Example 29 non-exportable \( n_1, n_2 \) remains unobservable even as we vary the top part of Fig. 4:

**Example 33.** Regarding Fig. 4 as the frame \( F_1 \), let \( L_0 \) be the locations below \( \{c_1, c_2\} \), and let \( \text{cut} = \{c_1, c_2\} \). Let \( F_2 \) contain \( L_0, \text{cut} \) as shown, and have any graph structure above cut such that cut remains a cut between the new structure and \( F_0 \). Let the new locations have any transition systems such that the local runs agree, i.e. cut-runs(\( F_2 \)) = cut-runs(\( F_1 \)). Then by Thm. 32, external inferences about \( \text{chans}(\{n_1, n_2\}) \) are guaranteed to blur out non-exportable events. ///

It is appealing that our security goal is independent of changes in the structure of the internet that we do not control. A similar property holds for the integrity goal of Example 29 as we alter the internal network. The converse questions—preserving the confidentiality property as the internal network changes, and the integrity property as the internet changes—appear to require a different, refinement-oriented theorem.

**Example 34.** Consider a frame \( F_1 \) representing a precinct, as shown in Fig. 2. It consists of a set of voters \( \pi = \{v_1, \ldots, v_k\} \), a ballot box \( BB_1 \), and a channel \( c_1 \) connecting that to the election commission \( EC \). The \( EC \) publishes the results over the channel \( p \) to the public \( Pub \).

We have proved that a particular implementation of \( BB_1 \) ensures that \( F_1 \) blurs the votes; we formalized this within the theorem prover PVS. That is, if a pattern of voting in precinct 1 is compatible with an observation at \( c_1 \), then any permutation of the votes at \( \pi \) is also compatible.

The cut-blur principle implies this blur also applies to observations at channel \( p \) to the public. Other implementations of \( BB_1 \) also achieve this property. ThreeBallot and VAV [42] appear to have this effect; they involve some additional data delivered to \( Pub \), namely the receipts for the ballots.1

However, elections generally concern many precincts. Frame \( F_2 \) contains \( i \) precincts, all connected to the election commission \( EC \) (Fig. 3). Taking \( L_0 = \pi \cup \{BB_1\} \), we may apply Thm. 32. We now have cut = \( \{c_1\} \). Thus, to infer that \( F_2 \) blurs observations of the voters in precinct 1, we need only check that \( \{c_1\} \) has no new local runs in \( F_2 \).

By symmetry, each precinct in \( F_2 \) enjoys the same blur. Thus—for a given local run at \( p \)—any permutation of the votes at \( \pi \) preserves compatibility in \( F_2 \), and any permutation of the votes at \( \pi \) preserves compatibility in \( F_2 \). However, Thm. 32 does not say that any pair of permutations at \( \pi \) and \( \pi \) must be jointly compatible. That is, does every permutation on \( \pi \) that respects the division between the precinct of the \( \pi \) and the precinct of the \( \pi \)'s preserve compatibility? Although

1Our claim is possibilistic. Quantitatively, this may no longer hold: Some permutations may be more likely than others, given the receipts [12, 38].

Thm. 32 does not answer this question, the answer is yes, as we can see by applying Lemma 41 to \( F_2 \).

Thm. 32 is a tool to justify abstractions. Fig. 4 is a sound abstraction of a variety of networks, and Fig. 2 is a sound abstraction of the various multiple precinct instances of Fig. 3.

**VIII. RELATING BLURS TO NONINTERFERENCE AND NONDEDUCIBILITY**

If we specialize frames to state machines (see Fig. 5), we can reproduce some of the traditional definitions. Let \( D = \{d_1, \ldots, d_k\} \) be a finite set of domains, i.e. sensitivity labels; \( \implies \subseteq D \times D \) specifies which domains are visible to others, and may influence them. We assume \( \implies \) is reflexive, though not necessarily transitive. \( A \) is a set of actions, and \( \text{dom}: A \rightarrow D \) assigns a domain to each action; \( O \) is a set of outputs.

\[ M = (S, s_0, A, \delta, \text{obs}) \]

is a (possibly non-deterministic) state machine with states \( S \), initial state \( s_0 \), transition relation \( \delta \subseteq S \times A \times S \), and observation function \( \text{obs}: S \times D \rightarrow O \). \( M \) has a set of traces, and each trace \( \alpha \) determines a sequence of observations for each domain [47], [54], [55].

\( M \) accepts commands from \( A \) along the incoming channels \( \epsilon_i^\text{in} \) from \( d_i \); each command \( a \in A \) received from \( d_i \) has sensitivity \( \text{dom}(a) = d_i \). \( M \) delivers observations along the outgoing channels \( \epsilon_i^\text{out} \). The frame requires a little extra memory, in addition to the states of \( M \), to deliver outputs over the channels \( \epsilon_i^\text{out} \).

\( F \) is star-like, since \( M \) holds an endpoint for each channel. Hence, if \( A \in \text{Exc}(F) \), all events in \( \text{proj}(A, M) \), and \( \leq_A \) is linearly ordered. Let us write:

\[ C_t = \{\epsilon_i^\text{in}, \epsilon_i^\text{out}\} \] for \( d_i \)'s input and output for \( M \);
\[ \text{vis}(d_i) = \{\epsilon_i^\text{in}: d_i \rightarrow d_i\} \] for inputs visible to \( d_i \);
\[ \text{IN} = \{\epsilon_i^\text{in}: 1 \leq x \leq k\} \] for the input channels;
\[ \text{input}(A) = A \upharpoonright \text{IN} \text{ for all input behavior in } A. \]

**Noninterference and nondeducibility.** Noninterference [20] and its variants are defined by purge functions \( p \) for each target domain \( d_i \), defined by recursion on input behaviors \( \text{input}(A) \). The original Goguen-Meseguer (GM) purge function \( p_\delta \) for \( d_i \) [20] retains the events \( e \in \text{input}(A) \) satisfying the predicate

\[ \text{chan}(e) \in \text{vis}(d_i). \]

A purge function for intransitive \( \rightarrow \) relations was subsequently proposed by Haigh and Young [23]. In the purge function for domain \( d_i \), any input event \( e_0 \in \text{input}(A) \) is retained if \( \text{input}(A) \) has an increasing subsequence \( e_0 \geq e_1 \geq \ldots \geq e_j \) where \( \text{dom}(\text{chan}(e_j)) = d_i \) and, for each \( k \) with \( 0 \leq k < j \),

\[ \text{chan}(e_k) \in \text{vis}(\text{dom}(e_{k+1})). \]

![Machine M, domains \{d_1, \ldots, d_k\}](image-url)
In [54], van der Meyden’s purge functions yield tree structures instead of subsequences; every path from a leaf to the root in these trees is a subsequence consisting of permissible effects chan(e_k) ∈ vis(dom(e_{k+1})). This tightens the notion of security, because the trees “forget” ordering information between events that lie on different branches to the root.

We formalize a purge function for a domain d_i ∈ D as being a function from executions A to some range set A. It should be sensitive only to input events in A (condition 1), and it should certainly reflect all the inputs visible to level d_i (condition 2). In most existing definitions, the range A consists of sequences of input events, though in van der Meyden’s [54], they are trees of input events. In [11], the range depends on how declassification conditions are defined.

Definition 35. Let F be as in Fig. 5, and A any set. A function p: Exc(F) → A is a d_i-purge function, where d_i ∈ D, iff
1. input(A) = input(A') implies p(A) = p(A');
2. p(A) = p(A') implies A ∪ vis(d_i) = A' ∪ vis(d_i).
If p is a d_i-purge, A ≈_p A' means p(A) = p(A').

Each purge p determines notions of noninterference and nondeducibility.

Definition 36. Let p be a purge function for d_i ∈ D. F is p-noninterfering, written F ∈ NI^p, iff, for all A, A' ∈ Exc(F),
\[ A ≈_p A' \implies A ∪ C_i = A' ∪ C_i. \]
F is p-nondeducible (F ∈ ND^p), iff, for all A, A' ∈ Exc(F),
\[ A ≈_p A' \implies A' ∪ IN ∈ J_{INAC_i}(A ∪ C_i). \]

Here we take non-deducibility to mean that d_i’s observations provide no more information about all inputs than the purge p preserves. Thus, A ∪ C_i is akin to Sutherland’s view [53, Sec. 5.2], although slightly adapted.

Sutherland’s hidden from appears to mean A' ∪ {e_i, d_j → d_i}, i.e. the inputs that would not be visible to d_i. This agrees with our proposed definition in the case Sutherland considered, namely the classic GM purge for noninterference. The assumption A ≈_p A' is meant to extend nondeducibility for other purges. As expected, noninterference is tighter than nondeducibility [53, Sec. 7].

Lemma 37. Let p be a purge function for domain d_i, F ∈ NI^p implies F ∈ ND^p.

Proof. Assume that F ∈ NI^p and A, A' ∈ Exc(F), where A ≈_p A'. By the definition, A ∪ C_i = A' ∪ C_i. Thus, J_{INAC_i}(A ∪ C_i) = J_{INAC_i}(A' ∪ C_i). But A' ∪ IN ∈ J_{INAC_i}(A' ∪ C_i), because A' is itself a witness.

NI^p and ND^p are not equivalent, as ND^p has an additional (implicit) existential quantifier. The witness execution showing that A' ∪ IN ∈ J_{INAC_i}(A ∪ C_i) may differ from A' on channels c \not∈ IN ∪ C_i, namely the output channels e_i^\text{out} for j \neq i.

The symmetry of nondisclosure (Lemma 12) does not hold for NI^p and ND^p. For instance, relative to the GM purge for flow to d_i, there may be noninterference for inputs at d_j, while there is interference for flow from d_i to d_j. The asymmetry arises because the events to be concealed are only inputs at the source, while the observed events are both inputs and outputs [53].

The idea of p-noninterference is useful only when M is deterministic, since otherwise the outputs observed on e_i^\text{out} may differ even when input(A) = input(A'). For nondeterministic M, nondeducibility is more natural.

Purges and blurs. We can associate a blur operator f^p with each purge function p, such that ND^p amounts to respecting the blur operator f^p. We regard ND^p as saying that the input/output events on C_i tell d_i no more about all the inputs than the purged input p(A) would disclose. We use a compatibility relation where the observed channels and the source channels overlap on e_i^\text{out}.

Definition 38. Let p be a purge function for d_i, and define the equivalence relation R ⊆ (IN-runs × IN-runs) by the condition:
\[ R(B_1, B_2) \text{ iff there exist } A_1, A_2 ∈ Exc(F) \text{ s.t.:} \]
\[ ( \bigwedge_{j=1,2} B_j = A_j ∪ IN ) ∧ A_1 ≈_p A_2. \]

Define f^p: P(IN-runs) → P(IN-runs) to close under the R-equivalence classes as in Lemma 22.

In fact, ND is a form of disclosure limited to within a blur:

Lemma 39. Let p be a purge function for domain d_i. For all F, F ∈ ND^p iff F f^p-limits IN-to-C_i flow.

Proof. 1. ND^p implies f^p-limited flow. Suppose that F ∈ ND^p; B_i ∈ C_i-runs; and B_i ∈ J_{INAC_i}(B_i). If B_i ∈ f^p(B_i), we must show that B_i ∈ J_{INAC_i}(B_i).

By Def. 38 there are A_1, A_2 such that
\[ A_1 ∪ IN = B_1, \quad A_2 ∪ IN = B_2, \quad A_1 ≈_p A_2. \]
Furthermore, let A witness B_i ∈ J_{INAC_i}(B_i). Then
\[ A' ∪ IN = B_i = A_1 ∪ IN. \]

So Def. 35, Clause 1 says A ≈_p A_1, and, by transitivity of ≈_p, also A ≈_p A_2. Since F ∈ ND^p,
\[ A_2 ∪ IN ∈ J_{INAC_i}(A ∪ C_i). \]
That is, B_2 ∈ J_{INAC_i}(B_i) as required.

2. f^p-limited flow implies ND^p. Assume J_{INAC_i}(B_i) is f^p-blurred for all B_i. We must show, for all A_1, A_2,
\[ A_1 ≈_p A_2 \implies (A_2 ∪ IN) ∈ J_{INAC_i}(A_1 ∪ C_i). \]

So choose executions with A_1 ≈_p A_2. By Def. 38, R(A_1 ∪ IN, A_2 ∪ IN), since A_1, A_2 satisfy the condition. Thus,
\[ A_2 ∪ IN ∈ J_{INAC_i}(A_1 ∪ C_i), \]
since J_{INAC_i}(A_1 ∪ C_i) is f^p-blurred and contains A_1 ∪ IN.

A frame F of this kind has definite inputs and outputs. The inputs are the events on IN, and the outputs are the
events on \( \text{OUT} = \{ c^\text{out}_x : 1 \leq x \leq k \} \). We may thus regard it as a function from inputs to outputs (or, if \( M \) is non-deterministic, to sets of outputs). In this context, one could compare blurs with the partial equivalence relation model or abstract noninterference [24], which apply only when the system is a function mapping inputs to outputs. One can also regard some \( d_j \) as using a strategy for future inputs on \( c^\text{in}_j \) based on current outputs on \( c^\text{out}_j \), recovering a form of nondeducibility on strategies [56].

Semantic sensitivity. Blur operators provide an explicit semantic representation of the information that will not be disclosed when flow is limited. This is in contrast to intransitive non-interference [47], [23], [54], which considers only whether the "\( \rightsquigarrow \) plumbing" among domains is correct.

**Example 40.** We represent Imaginary Weather Forecasting (IWF, see Example 24) as a state machine frame as in Fig. 5. It has domains \( \{ ws, \ell, p, cmp \} \) for the weather service, low-tier customer, premium-tier customer and compression service respectively. Let \( \rightsquigarrow \) be the smallest reflexive (but intransitive) relation extending Eqn. 4, where all reports must flow through the compression service:

\[

ws \rightsquigarrow cmp \rightsquigarrow p \text{ and } cmp \rightsquigarrow \ell. \tag{4}

\]

The \( cmp \) service should compress reports lossily before sending them to \( \ell \) and compress them losslessly for \( p \). However, a faulty \( cmp \) may compress losslessly for both \( \ell \) and \( p \). Purge functions [23], [47], [54] do not distinguish between correct and faulty \( cmps \). In both cases, all information from \( ws \) does indeed pass through \( cmp \). The blur of Example 24, however, defines the desired goal semantically. With the faulty \( cmp \), the high-resolution data compatible with the observation of \( \ell \) is more sharply defined than an \( f \)-blurred set. ///

IX. Future Work

We have explored how the graph structure of a distributed system helps to constrain information flow. We have established the cut-blur principle. It allows us to propagate conclusions about limited disclosure from a cut set cut to more remote parts of the graph. These ideas are much more widely applicable than the simple examples that we have used here.

Quantitative treatment. It should be possible to equip frames with a quantitative information flow semantics. One obstacle here is that our execution model mixes some choices which are natural to view probabilistically—for instance, selection between different outputs when both are permitted by an LTS—with others that seem non-deterministic. The choice between receiving an input and emitting an output is an example of this, as is the choice between receiving inputs on different channels. This problem has been studied (e.g. [7], [8]), but a tractable semantics may require new ideas.

A Dynamic Model. Instead of building \( \text{ends}(\ell) \) into the frame, so that it remains fixed through execution, we may alternatively regard it as a component of the states of the individual locations. Let us regard \( \text{traces}(\ell) \) as generated by a labeled transition system \( \text{LTS}(\ell) \). Then we may enrich the labels \( c, v \) so that they also involve a sequence of endpoints \( \pi \subseteq \mathcal{E} \) :

\[

(c, v, \pi).
\]

The transition relation of \( \text{LTS}(\ell) \) is then constrained to allow a transmission \( (c, v, \pi) \) in a state only if \( \pi \subseteq \text{ends}(\ell) \) holds in that state, in which case \( \pi \) is omitted in the next state. A reception \( (c, v, \pi) \) causes \( \pi \) to be added to the next state of the receiving location.

The cut-blur principle remains true in an important case: A set cut is an invariant cut between src and obs if it is an undirected cut, and moreover the execution of the frame preserves this property. Then the cut-blur principle holds in the dynamic model for invariant cuts.

This dynamic model suggests an analysis of security-aware software using object capabilities. Object capabilities may be viewed as endpoints entry\((c)\). To use it, one sends a message to the object itself, which holds exit\((c)\). To transfer a capability, one sends entry\((c)\) over some \( c' \).

McCamant and Ernst [34]'s quantitative approach generates a directed graph of this sort in memory at runtime. Providing a maximum over all possible runs would appear to depend on inferring some invariants on the structure of the graphs. Our methods might be helpful for this.

Cryptographic Masking. Encryption is not a blur. Encrypting messages makes their contents unavailable in locations lacking the decryption keys. In particular, locations lacking the decryption key may form a cut set between the source and destination of the encrypted message. However, at the destinations, where the keys are available, the messages can be decrypted and their contents observed. Thus, the cut-blur theorem implies it would be wrong to view encryption as a blur in this set-up: Its effects can be undone beyond the cut.

Several approaches are possible here. We would like to use the resulting set-up to reason about cryptographic voting systems, such as Helios and Prêt-à-Voter [2], [49].

We also intend to provide tool support for defining relevant blurs and establishing that they limit disclosure in several application areas.

Acknowledgments. We are grateful to Megumi Ando, Aslan Askarov, Stephen Chong, John Ramsdell, and Mitchell Wand. In particular, John Ramsdell formalized Thms. 28 and 32 in PVS, as well as Example 26; this suggested Example 34.

REFERENCES


APPENDIX

We gather here additional lemmas, and a few more proofs.

Lemma 13.
1. Suppose $C_0 \subseteq C_1$ and $C'_0 \subseteq C'_1$. If $F$ has no disclosure from $C_1$ to $C'_1$, then $F$ has no disclosure from $C_0$ to $C'_0$.
2. When $C_1, C_2, C_3 \subseteq \mathcal{C}_H$,
   $$J_{C_3 \setminus C_1}(B_1) \subseteq \bigcup_{B_2 \in J_{C_3 \setminus C_2}(B_2)} J_{C_3 \setminus C_2}(B_2).$$

Proof. 1. Suppose $B_0$ is a $C_0$-run, and $B'_0$ is a $C'_0$-run. We want to show that $B'_0 \in J_{C'_3 \setminus C'_0}(B')_1$.

Since they are local runs, there exist $A_0, A'_0 \in \text{Exc}(F)$ such that $B_0 = A_0 \upharpoonright C_0$ and $B'_0 = A'_0 \upharpoonright C'_0$. But let $B_1 = A_0 \upharpoonright C_1$ and let $B'_1 = A'_0 \upharpoonright C'_1$. By no-disclosure, $B'_1 \in J_{C'_3 \setminus C'_1}(B_1)$. So there is an $A \in \text{Exc}(F)$ such that $B_1 = A \upharpoonright C_1$ and $B'_1 = A \upharpoonright C'_1$.

However, then $A$ witnesses for $B'_0 \in J_{C'_3 \setminus C'_0}(B')_1$: After all, since $C_0 \subseteq C_1$, $A \upharpoonright C_0 = (A \upharpoonright C_1) \upharpoonright C_0$. Similarly for the primed versions.

2. Suppose that $B_3 \in J_{C_3 \setminus C_1}(B_1)$, so that there exists an $A \in \text{Exc}(F)$ such that $B_3 = A \upharpoonright C_1$ and $B_3 = A \upharpoonright C_3$. Letting $B_2 = A \upharpoonright C_2$, the execution $A$ ensures that $B_2 \in J_{C_3 \setminus C_2}(B_1)$ and $B_3 \in J_{C_3 \setminus C_2}(B_2)$. $\square$

We now consider different frames $F_1, F_2$ that overlap on a common subset $L_0$, and show how local runs in the two can be pieced together. In this context, we use the notation of Def. 30, such as left$_0$ for channels between locations $L_0$ shared between $F_1$ and $F_2$, cut$_0$ for the set of channels forming the boundary, and right$_0$ for the channels unattached to $L_0$ in $F_1$.

Lemma 41. Let $L_0$ be shared between frames $F_1, F_2$. Let $B_{lc} \in \text{(left$\cup$cut)-runs}_1$ and $B_{rc} \in \text{(right$\cup$cut)-runs}_2$ agree on cut, i.e. $B_{lc} \upharpoonright cut = B_{rc} \upharpoonright cut$. Then there is an $A \in \text{Exc}(F_2)$ such that

$$B_{lc} = A \upharpoonright (\text{left$\cup$cut}) \text{ and } B_{rc} = A \upharpoonright (\text{right$\cup$cut}).$$

Proof. Since $B_{lc}$ and $B_{rc}$ are local runs of $F_1, F_2$ resp., they are restrictions of executions, so choose $A_1 \in \text{Exc}(F_1)$ and $A_2 \in \text{Exc}(F_2)$ so that $B_{lc} = A_1 \upharpoonright (\text{left$\cup$cut})$ and $B_{rc} = A_2 \upharpoonright (\text{right$\cup$cut})$. Now define $A$ by stipulating:

$$ev(A) = ev(B_{lc}) \cup ev(B_{rc}) \quad (5)$$

$$\preceq_A = \text{ the least partial order extending } \preceq_{B_{lc}} \cup \preceq_{B_{rc}}. \quad (6)$$

Since $A_1, A_2$ agree on cut, $ev(A) = ev(B_{lc} \upharpoonright \text{left}) \cup ev(B_{rc})$, and we could have used the latter as an alternate definition of $ev(A)$, as well as the symmetric restriction of $B_{rc}$ to right$_0$ leaving $B_{lc}$ whole.

The definition of $\preceq_A$ as a partial order is sound, because there are no cycles in the union (6). Cycles would require $A_1$ and $A_2$ to disagree on the order of events in their restrictions to cut, contrary to assumption. Likewise, the finite-predecessor property is preserved: $x_0 \preceq_A x_1$ iff $x_0, x_1$ belong to the same $B_{rc}$ and are ordered there, or else there is an event in $B_{rc} \upharpoonright cut$ which comes between them. So the events preceding $x_1$ form the finite union of finite sets. Thus, $A \in \text{ES}(F_2)$.

Moreover, $A$ is an execution $A \in \text{Exc}(F_2)$: If $\ell \in L_0$, then $\text{proj}(A, \ell) = \text{proj}(B_{lc}, \ell)$, and the latter is a trace in $\text{traces}_1(\ell) = \text{traces}_2(\ell)$. If $\ell \notin L_0$, then $\text{proj}(A, \ell) = \text{proj}(B_{rc}, \ell)$, and the latter is a trace in $\text{traces}_2(\ell)$.

There is no $\ell$ with channels in both left and right$_0$. $\square$

What makes this proof work? Any one location either has all of its channels lying in left$_0 \cup$cut$_0$ or else all of them lying in right$_0 \cup$cut. When piecing together the two executions $A_1, A_2$ into a single execution $A$, no location needs to be able to execute a trace that comes partly from $A_1$ and partly from $A_2$. This is what determines our definition of cuts using the undirected graph $\text{ungr}(F)$.

We now prove the two-frame analog of Lemma 15.

Lemma 31. Let $L_0$ be shared between frames $F_1, F_2$. Let $\text{src} \subseteq \text{left}$, and $B_0 \in \text{cut$_0$-runs}_1 \cap \text{cut$_0$-runs}_2$.

1. $J^1_{\text{src$\cup$cut}}(B_0) = J^1_{\text{cut$_0$-runs}}(B_0)$.

2. Assume $\text{cut$_0$-runs}(F_2) \subseteq \text{cut$_0$-runs}(F_1)$. Let $\text{obs} \subseteq \text{right$_0$}$, and $B_0 \in \text{obs-runs}_2$. Then

$$J^2_{\text{src$\cup$cut}}(B_0) = \bigcup_{B_0 \in J^1_{\text{src$\cup$cut}}(B_0)} J^2_{\text{src$\cup$cut}}(B_0).$$

Proof. 1. First, we show that $B_0 \in J^1_{\text{src$\cup$cut}}(B_0)$ implies $B_0 \in J^1_{\text{cut$_0$-runs}}(B_0)$.

Let $A_1$ witness for $B_0 \in J^1_{\text{src$\cup$cut}}(B_0)$, and let $A_2$ witness for $B_0 \in \text{cut$_0$-runs}_2$. Define

$$B_{lc} = A_1 \upharpoonright (\text{left$\cup$cut}) \text{ and } B_{rc} = A_2 \upharpoonright (\text{right$_0$$\cup$cut}).$$

Now the assumptions for Lemma 41 are satisfied. So let $A \in \text{Exc}(F_2)$ restrict to $B_{lc}$ and $B_{rc}$ as in the conclusion. Thus, $A \upharpoonright \text{src} = B_0$.

For the converse, we rely on the symmetry of “$L_0$ is shared between frames $F_1, F_2$.”

2. By the assumption, whenever $B_0 \in J^2_{\text{cut$_0$-runs}}(B_0)$, then also $B_0 \in \text{cut$_0$-runs}_1$. Thus, we can apply part 1 after using Lemma 13:

$$J^2_{\text{src$\cup$cut}}(B_0) \subseteq \bigcup_{B_0 \in J^1_{\text{cut$_0$-runs}}(B_0)} J^2_{\text{src$\cup$cut}}(B_0).$$

For the reverse inclusion, assume that $B_0 \in J^1_{\text{cut$_0$-runs}}(B_0)$, where $B_0 \in J^2_{\text{cut$_0$-runs}}(B_0)$. Thus, we can apply Lemma 41, obtaining $A \in \text{Exc}(F_2)$ which agrees with $B_0, B_{lc}$, and $B_{rc}$. $A$ witnesses for $B_0 \in J^2_{\text{cut$_0$-runs}}(B_0)$.

We now turn to the one-frame corollary, which we presented earlier as Lemma 15.
Lemma 15. Let cut be an undirected cut between src, obs, and let $B_0 \in$ src-runs. Then

$$J_{\text{src-obs}}(B_o) = \bigcup_{B_c \in J_{\text{cut-obs}}(B_o)} J_{\text{src-cut}}(B_c).$$

Proof. Define $L_0$ to be the smallest set of locations such that

1. $\ell \in L_0$ if $\text{chans}(\ell) \cap \text{src} \neq \emptyset$;
2. $L_0$ is closed under reachability by paths that do not traverse cut.

$L_0$ is shared between $\mathcal{F}$ and itself. Moreover, for the set of $\ell \in L_0 \subseteq \mathcal{L} \subseteq A$ that actually lies on the boundary of $A|\text{src}$, $B$ is closed under reachability by paths that do not cut.

By Lemma 30, we have

$$J_{\text{src-obs}}(B_o) = \bigcup_{B_c \in J_{\text{cut-obs}}(B_o)} J_{\text{src-cut}}(B_c).$$

Since $\text{cut}_0 \subseteq \text{cut}$,

$$\bigcup_{B_c \in J_{\text{cut-obs}}(B_o)} J_{\text{src-cut}}(B_c) \subseteq \bigcup_{B_c \in J_{\text{cut-obs}}(B_o)} J_{\text{src-cut}}(B_c).$$

For the converse, suppose that $B_s \in J_{\text{src-cut}}(B_c)$, for $B_c \in J_{\text{cut-obs}}(B_o)$. Then there is $A$ such that $A \upharpoonright \text{src} = B_s$ and $A \upharpoonright \text{obs} = B_o$. Thus, $B_s \in J_{\text{src-cut}}(A \upharpoonright \text{cut}_0)$ and $A \upharpoonright \text{cut}_0 \in J_{\text{cut-obs}}(B_o)$. □

The cut-blur principle is also the one-frame corollary of Thm. 32. The proofs are very similar.

Theorem 32. Suppose that $L_0$ is shared between frames $\mathcal{F}_1, \mathcal{F}_2$, and assume $\text{cut-runs}(\mathcal{F}_2) \subseteq \text{cut-runs}(\mathcal{F}_1)$. Consider any src $\subseteq$ left and obs $\subseteq$ right. If $\mathcal{F}_1$ $f$-limits src-to-cut flow, then $\mathcal{F}_2$ $f$-limits src-to-obs flow.

Proof. By the hypothesis, $f$ is a blur operator. Letting $B_o \in$ obs-runs, we want to show that $J_{\text{src-obs}}^2(B_o)$ is an $f$-blurred set, i.e. $J_{\text{src-obs}}^2(B_o) = f(J_{\text{src-obs}}^1(B_o))$.

For convenience, let $S_c = J_{\text{cut-obs}}^2(B_o)$. By Lemma 31,

$$J_{\text{src-obs}}^2(B_o) = \bigcup_{B_c \in S_c} J_{\text{src-cut}}^1(B_c);$$

thus, we must show that the latter is $f$-blurred. By the assumption that each $J_{\text{src-cut}}^1(B_c)$ is $f$-blurred, we have $J_{\text{src-cut}}^1(B_c) = f(J_{\text{src-cut}}^1(B_c))$. Using this and the union property (Eqn. 1):

$$\bigcup_{B_c \in S_c} J_{\text{src-cut}}^1(B_c) = \bigcup_{B_c \in S_c} f(J_{\text{src-cut}}^1(B_c)) = f(\bigcup_{B_c \in S_c} J_{\text{src-cut}}^1(B_c)).$$

Hence, $J_{\text{src-obs}}^2(B_o)$ is $f$-blurred. □