Many-to-Many Information Flow Policies

Paolo Baldan, Alessandro Beggiato, Alberto Lluch Lafuente

Many-to-Many Information Flow Policies

Paolo Baldan[1], Alessandro Beggiato[2], and Alberto Lluch Lafuente[3]

[1] Università di Padova, Dipartimento di Matematica,
baldan@math.unipd.it
[2] IMT School for Advanced Studies Lucca
alessandro.beggiato@imtlucca.it
[3] Technical University of Denmark, DTU Compute
albl@dtu.dk

Abstract. Information flow techniques typically classify information according to suitable security levels and enforce policies that are based on binary relations between individual levels, e.g., stating that information is allowed to flow from one level to another. We argue that some information flow properties of interest naturally require coordination patterns that involve *sets* of security levels rather than individual levels: some secret information could be safely disclosed to a set of confidential channels of incomparable security levels, with individual leaks considered instead illegal; a group of competing agencies might agree to disclose their secrets, with individual disclosures being undesired, etc. Motivated by this we propose a simple language for expressing information flow policies where the usual admitted flow relation between individual security levels is replaced by a relation between sets of security levels, thus allowing to capture coordinated flows of information. The flow of information is expressed in terms of causal dependencies and the satisfaction of a policy is defined with respect to an event structure that is assumed to capture the causal structure of system computations. We suggest applications to secret exchange protocols, program security and security architectures, and discuss the relation to classic notions of information flow control.

Keywords: Information Flow, Coordination, Concurrency, Declassification, Non-Interference, Causality, Event Structures.

1 Introduction

As the number of interconnected devices increases, the focus on security-related aspects of coordinated computations gains more and more relevance and appeal. Techniques for controlling and enforcing the flow of information need to be applied, and possibly extended to deal with coordination aspects. Typically, the entities of a system are assigned a security level, and information flow policies prescribe which interactions are legal and which are forbidden. This is normally expressed via a relation that models the admitted flows between security levels.

Motivation and Problem Statement. The information flow relations used in the literature to model policies are almost invariably binary relations between individual security levels. This paper is motivated by the observation that some desired information flow properties naturally involve suitable coordinated *sets* of security levels rather than mere individual levels.

For example, some secret information (say, owned by a government agency E, cf. Figure 1) could be safely disclosed to a set of confidential channels of incomparable security levels (say, corresponding to competing investors C and D) *simultaneously*, with individual leaks considered instead illegal or unfair. This is for instance, the spirit of U.S. security and exchange commission's *regulation fair disclosure* [22]. Dually, a group of competing companies (say A and B in Figure 1) may agree to *collectively* disclose their secrets (say to the government agency E), with individual disclosures being undesired. This paper is motivated by such scenarios and, in general, by the following question: *what is a natural notion of information flow policies that regulate flows among sets of security levels?*

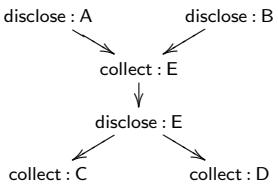


Fig. 1. Information flow example.

Contributions. We address the question by proposing a simple policy specification language that extends the usual security diagrams by allowing relations between *sets* of security levels instead of just *single* levels. The clauses in our policies are of the form $\mathsf{A}_1, \ldots, \mathsf{A}_m \rightsquigarrow \mathsf{B}_1, \ldots, \mathsf{B}_n$, intuitively meaning that the security levels $\mathsf{A}_1, \ldots, \mathsf{A}_m$ are allowed to coordinate in order to let information flow to security levels $\mathsf{B}_1, \ldots, \mathsf{B}_n$.

In our approach the flow of information between entities is captured in terms of the existence of causal dependencies between events representing occurrences of actions of such entities. In particular, we use event structures [16,25] as a reference semantic model. The idea is that causal dependencies between events represent the transfer of some information. Thus causal dependencies are required to obey to coordination patterns as prescribed by the information flow policy. For traditional intransitive binary policies any flow of information, i.e., any (direct) causality $a < b$ between events a and b needs to be allowed by the policy, i.e., if the level of a is A and the level of b is B then the policy has to include a clause $\mathsf{A} \rightsquigarrow \mathsf{B}$. We generalise this to many-to-many policies by requiring that any direct causality $a < b$ is part of a possibly more complex interaction that conforms to a coordination pattern allowed by the policy, i.e., if A_1 and B_1 are the security levels of a and b, respectively, there must exist a clause $\mathsf{A}_1, \mathsf{A}_2 \ldots \mathsf{A}_n \rightsquigarrow \mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_m$ in the policy and events $a_1, a_2, \ldots, a_n$, $b_1, b_2, \ldots, b_m$ (with b equal to some b_k) such that each event a_i has level A_i, each event b_j has level B_j, and events $a_1, \ldots a_n$ are (suitably coordinated) causes of the events $b_1, \ldots, b_m$.

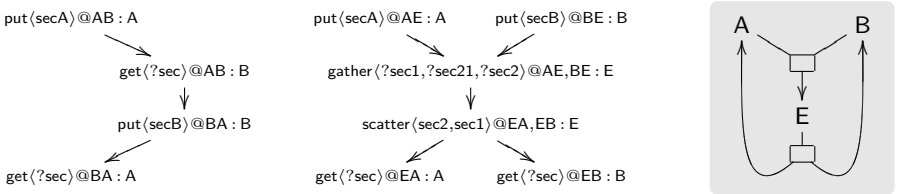
As an example, consider the diagram of Figure 1, where arrows represent direct causalities between events and the security levels coincide with the principals. For events we use the notation name : level. The direct causality from event disclose : A to event collect : E is allowed by the policy A, B $\rightsquigarrow$ E since collect : E is also causally dependent on disclose : B, thus providing some guarantee of the fact that A and B disclose their secrets collectively. Analogously, the direct causality from disclose : E to collect : C is allowed by the policy E $\rightsquigarrow$ A, B since there is a causality relation from disclose : E to collect : D as well, yielding some sort of simultaneity in the disclosure of the secrets from E to C, D.

We study several properties of our policy language. In particular, we observe that checking whether a system satisfies a policy is decidable for a general class of event structures, the so-called regular trace event structures [24]. As a matter of fact, policy satisfaction is expressible as a first-order property and the corresponding model checking problem is decidable [12]. We also discuss the relation with classical notions of information flow control, including non-interference and declassification, and suggest applications beyond secret exchange protocols, including program security and routing in security architectures.

Synopsis. Section 2 introduces several motivating examples, including a running example that is used throughout the paper. Section 3 provides some technical background on event structures. Section 4 presents the policy language, the notion of policy satisfaction and a decidability result. Section 5 compares with notions of information flow based on interleaving semantics, in particular with trace- and bisimulation-based non-interference. Section 6 discusses other related works. Section 7 concludes our paper and outlines future research.

2 Motivating Examples

We introduce here some examples that have motivated our work and that we envisage as application domains for many-to-many information flow analysis.

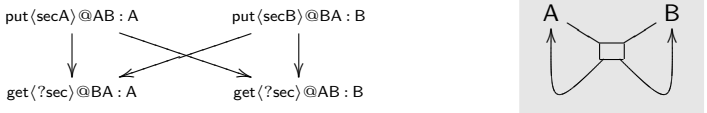


(a) Protocol P_1 (b) Protocol P_2 (c) Policy

Fig. 2. Two secret exchange protocols and a security policy.

Simultaneous Secret Exchange. Consider first the problem of exchanging a secret between two parties, say Alice and Bob. We shall use this as a running example throughout the paper. A way to solve the problem would be to proceed according to the protocol in Figure 2a, where Alice starts sending her secret to Bob, which then replies with his own secret. The graphical representation (technically a security-labelled event structure) will be explained later. Here it suffices to understand that the figure represents the structure of the communication in an execution, where an event $\mathsf{put}\langle \mathsf{m}\rangle@\mathsf{C} : \mathsf{P}$, represents party P sending m on channel C, and an event $\mathsf{get}\langle \mathsf{t}\rangle@\mathsf{C} : \mathsf{P}$, represents party P receiving a message from channel C to be saved according to the pattern t (binding/formal fields being denoted with a leading question mark "?"). Arrows represent (direct) causal dependencies between events. The protocol has essentially the same structure of classical key exchange protocols [14] and does not solve one of the main concerns of the so-called *simultaneous secret exchange problem*, which is to avoid or minimise competitive advantage among the parties exchanging the secrets (see e.g. [17]). If we assume to deal with information of two security levels (one for each party), a standard approach to information flow does not help much in this scenario, as we can just allow flows between Alice and Bob (and thus accept any protocol with no guarantee) or forbid them (and thus reject all protocols).

One standard solution to the simultaneous secret exchange problem is to use an intermediary (say Eve). Many-to-many information flow policies can be used to specify some desired properties of intermediary-based protocols. For example, we may require that the intermediary forwards information to Alice and Bob simultaneously (denoted by an information flow policy $\mathsf{Eve} \rightsquigarrow \mathsf{Alice}, \mathsf{Bob}$), and that the intermediary accepts information from Alice and Bob only if collectively disclosed (denoted by an information flow policy by $\mathsf{Alice}, \mathsf{Bob} \rightsquigarrow \mathsf{Eve}$). A graphical representation for this security policy, that will be refined and explained in detail later, can be found in Figure 2c. The protocol sketched in Figure 2b, which uses multi-party interactions, satisfies the desired information flow properties. The protocol uses in particular an MPI-like gather operation to (point-wise) collect a list messages from different sources, and MPI-like scatter operation to piece-wise broadcast the list of secrets.



(a) Protocol P_3

(b) Policy

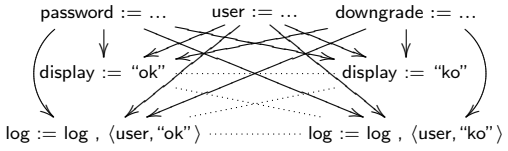
Fig. 3. A secret exchange protocol satisfying a policy without intermediary.

We shall see that the causality approach to information flow allows one to provide stronger guarantees on secret exchange protocols, by only admitting in-

formation to flow collectively and simultaneously between Alice and Bob, namely, Alice, Bob $\rightsquigarrow$ Alice, Bob (see Figure 3b, for a graphical representation), even without intermediaries. This is realised by the protocol in Figure 3a, where vertical and cross dependencies are control and data dependencies, respectively.

Language-Based Security and Declassification. We use a classic problem of declassification in access control to illustrate how our approach can be used to check information flow properties in the traditional sense [20], and how our policies can be used to specify several useful forms of declassification [13,21]. The access control program below, on the left, written in a simple imperative programming language in the style of [20], is checking a password and preparing a reply to the user in a variable display. A causal semantics to the program can be given based on the following idea: (i) events correspond to variable updates or initialisations, (ii) an update x := e causally depends on previous updates of all variables in e or their initialisation, (iii) if a control point is conditioned by y then all updates in all branches of the control causally depend on the latest update of y (or its initialisation), and (iv) conflict relations (represented as dotted lines) capture branching. For the program at hand, the resulting event structure can be found on the right.

```
if downgrade then
    if check(user,password) then
        display := "ok";
        log := log , ⟨user,"ok"⟩;
    else
        display :="ko";
        log :=log , ⟨user,"ko"⟩;
```



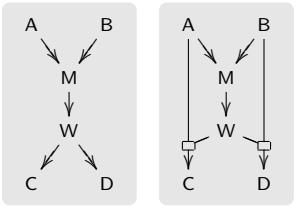
Disregarding whether the password is correct or not, there is a flow of information concerning the password to the user, represented as a causality relation from the latest update of the password to the updates of variable display.

For simplicity assume that each variable has its own security level, coinciding with the variable name. A standard security type system would consider the program to be insecure and our approach would agree on that in absence of a policy password $\rightsquigarrow$ display allowing the leaks. Of course, such a policy is not desirable in general (e.g. the user may be malicious). However, the program provides some guarantees that make it acceptable from the security point of view. First, the reply is also influenced by a check on variable downgrade, which may be used to disable login (e.g. after several unsuccessful attempts). This provides a standard form of controlled declassification. Requiring that the declassification is controlled as desired can be done through a policy password, downgrade $\rightsquigarrow$ display. In addition, the program above is using a log to keep track of logging attempts. This provides the additional desirable property that password leaks are only possible when the information concerning the access attempt also flows to the log (denoted user, password $\rightsquigarrow$ display, log).

Security architectures. A third motivating case are systems where information of different security levels needs to traverse shared resources, like a routing network. An archetypal example is the *Multiple Independent Levels of Security* (MILS)

architecture [19]. The diagram on the left of the figure below depicts a simplified version of such an architecture, inspired by the case study of [11]. Information from security level A (resp. B) should be allowed to flow into C (resp. D) only. Messages are routed through a common network made of a multiplexer component (M) that accepts messages from both A and B and forwards it to a demultiplexer component (W), which dispatches the messages either to C or to D, according to routing information.

The diagram on the left of the figure can be also seen as an information flow policy (isomorphic to the architecture) aiming at enforcing the desired flows. The problem of this naive policy is that it allows also for undesired flows, e.g., from A to D and from B to C, indirectly through the network: a protocol wrongly routing the information would be admitted by the policy. Consider instead a policy where individual flows from W to C and D are not allowed, and instead, they need to be collectively produced with A and B, respectively (denoted $\mathsf{A},\mathsf{W} \rightsquigarrow \mathsf{C}$ and $\mathsf{B},\mathsf{W} \rightsquigarrow \mathsf{D}$). The new policy (sketched on the right of the figure) would reject protocols wrongly routing the messages.



3 Event Structures

We model the causal behaviour of a system with prime event structures [16,25], a well-studied semantic model of concurrency where causality is a primitive notion. The way in which an event structure is associated to a specific formalism will depend on the system features one intends to capture (data/control flow, etc.).

Definition 1 (event structure). *An* event structure $\mathcal{E} = \langle E, \leq, \# \rangle$ *consists of a set E of events, a partial order relation $\leq \subseteq E \times E$ called* causality*, and an irreflexive symmetric relation $\# \subseteq E \times E$ called* conflict *such that:*

(i) for all $e \in E$ the set of causes $[e] = \{e' \in E \mid e' \leq e\}$ is finite (finitariness);
(ii) for all $e, e', e'' \in E$, if $e \# e'$ and $e' \leq e''$ then $e \# e''$ (conflict inheritance).

For $e, e' \in E$ we write $e < e'$ (e is a proper cause of e') if $e \leq e'$ and $e \neq e'$. We say that e is a direct cause of e' (denoted $e \lessdot e'$) if $e < e'$ and for all $e'' \in E$ if $e \leq e'' \leq e'$ either $e'' = e$ or $e'' = e'$. The causes of a set of events $X \subseteq E$ are defined as $[X] = \bigcup_{e \in X} [e]$. We lift causality to sets of events, with a universal interpretation, i.e., for $X, Y \subseteq E$, we write $X < Y$ (resp. $X \lessdot Y$) if for all $e \in X$ and $e' \in Y$ we have $e < e'$ (resp. $e \lessdot e'$). For $e \in E$ we define the set of its *proper causes* as $(e) = [e] \setminus \{e\}$. We say that $e, e' \in E$ are in *direct conflict*, written $e \#_\mu e'$, when $e \# e'$ and for all $e'' \in (e)$ it holds $\neg(e'' \# e')$ and for all $e''' \in (e')$ it holds $\neg(e \# e''')$, i.e., the conflict is not inherited.

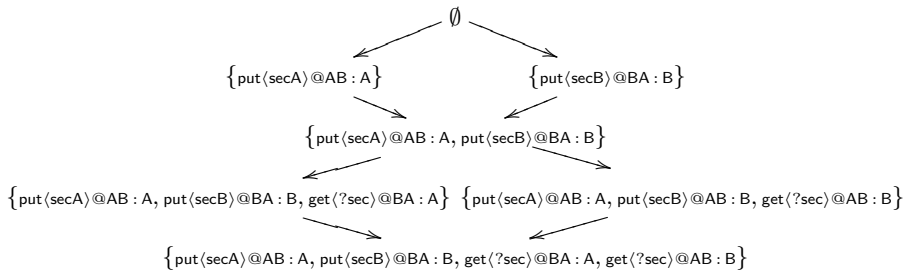


Fig. 4. Configurations of the event structure in Figure 3a.

Figures 2a–3a show three event structures corresponding to different protocols in our running example. The set of events correspond to communication operations, and are annotated with the initials of the principal executing the action (which is actually the security level assigned to the event, as we shall explain later). Causality is represented graphically by directed arrows, while conflict (to be seen in subsequent examples) is represented by dotted undirected lines. For the sake of a lighter notation, we follow the tradition of depicting direct causalities and conflicts only.

Event structures describe the possible events in computations and their mutual relations. When $e < e'$ then e must necessarily occur before e', while if $e \# e'$ then the occurrence of e excludes e' (and vice versa). Computations are characterised as conflict-free sets of events, containing the causes of all events.

Definition 2 (configuration). *Let $\mathcal{E} = \langle E, \leq, \# \rangle$ be an event structure. A* configuration *of $\mathcal{E}$ is a subset $C \subseteq E$ such that $\neg(e \# e')$ for all $e, e' \in C$ and $[C] = C$. The set of configurations of $\mathcal{E}$ is denoted $\mathcal{C}(\mathcal{E})$.*

For any event e, the causes $[e]$ and the proper causes (e) are configurations.

Figure 4 depicts all the configurations of the event structure of Figure 3a. They are related by arrows that represent transitions, i.e., how a configuration can be extended to another configuration by adding an event.

Definition 3 (extension). *Let $\mathcal{E} = \langle E, \leq, \# \rangle$ be an event structure, and let $C \in \mathcal{C}(\mathcal{E})$. We say that an event e extends C when $e \notin C$ and $C \cup \{e\} \in \mathcal{C}(\mathcal{E})$. In this situation we write $C \oplus e$ for $C \cup \{e\}$. The set of possible extensions of a configuration C is $\mathsf{PE}(C) = \{e \in E \mid C \oplus e \in \mathcal{C}(\mathcal{E})\}$.*

The transition system semantics of an event structure is obtained considering configurations as states and extensions as transitions.

Definition 4 (transition system). *The* transition system *of an event structure $\mathcal{E} = \langle E, \leq, \# \rangle$ is the tuple $TS(\mathcal{E}) = \langle \mathcal{C}(\mathcal{E}), \{C \to C \oplus e \mid C \in \mathcal{C}(\mathcal{E}) \wedge e \in E\} \rangle$.*

The diagram of Figure 4 represents the transition system of the event structure in Figure 3a. When events carry a label, the above definition yields labelled transition systems with transitions $C \xrightarrow{a} C \oplus e$, where a is the label of e.

4 Many-to-Many Information Flow Policies

We introduce here our notion of many-to-many information flow policies, which describe the legal interactions among sets of security levels. Section 4.1 presents the policy language, Section 4.2 defines the semantics, Section 4.3 studies some of its properties, and Section 4.4 provides a decidability result.

4.1 A Policy Language for Many-to-Many Information Flows

We start by introducing the syntax of many-to-many information flow policies.

Definition 5 (many-to-many information flow policy). *Let $\mathscr{L}$ be a finite set of security levels and $\mathscr{C} = \{d, f\}$ be a set of coordination constraints. A* many-to-many multi-level information flow policy *is a pair $\Pi = \langle \mathscr{L}, \rightsquigarrow \rangle$ where $\rightsquigarrow \subseteq 2^{\mathscr{L}} \times 2^{\mathscr{C}} \times 2^{\mathscr{L}}$ and $(\mathsf{A}, \emptyset, \mathsf{A}) \in \rightsquigarrow$ for all $\mathsf{A} \in \mathscr{L}$. We denote by $\mathcal{P}$ the set of all policies.*

We will write $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ for $(\mathbb{A}, \sigma, \mathbb{B}) \in \rightsquigarrow$, often dropping brackets from σ, $\mathbb{A}$ and $\mathbb{B}$. Security levels in $\mathscr{L}$ are ranged over by $\mathsf{A}, \mathsf{B}, \ldots$ and sets of security levels are ranged over by $\mathbb{A}, \mathbb{B}, \ldots$. The requirement that $\mathsf{A} \rightsquigarrow \mathsf{A}$ for all $\mathsf{A} \in \mathscr{L}$ is natural in information flow: information is always allowed to flow within the same level of security. Finiteness of the set of security levels $\mathscr{L}$ is a natural assumption. It could be meaningful to allow $\mathscr{L}$ to be infinite only when security levels can be generated dynamically. The theory in the paper could be adapted trivially apart from the decidability result in Section 4.4, that relies on regularity of the model which in turn implies the finiteness of $\mathscr{L}$.

Note that an information flow policy can be seen as a directed hyper-graph whose hyper-arcs are labelled by (possibly empty) subsets of the alphabet of coordination constraints $\mathscr{C}$ (to be explained later). This analogy is exploited in the visual representation of security policies.

Informally, a clause $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ specifies that a group of entities whose set of security levels is $\mathbb{A}$ is allowed to influence a group of entities whose set of security levels is $\mathbb{B}$, subject to the coordination constraints in σ. E.g., a policy

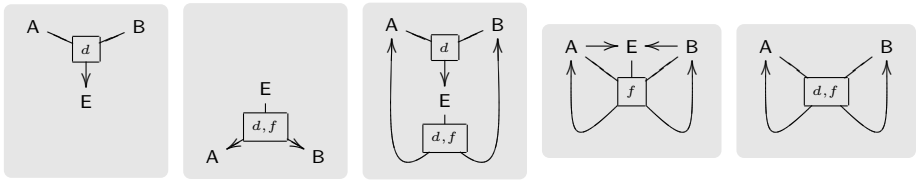
$$\mathsf{Alice}, \mathsf{Bob} \rightsquigarrow \mathsf{Eve} \tag{1}$$

allows Alice and Bob to influence Eve, while the policy

$$\mathsf{Eve} \rightsquigarrow \mathsf{Alice}, \mathsf{Bob} \tag{2}$$

allows Eve to influence Alice and Bob. We will see that the above two policies can be combined to allow Alice and Bob to exchange their secrets using Eve as an intermediary, and providing some fairness guarantees. A different policy that would allow a similar flow of information is

$$\begin{aligned} \mathsf{Alice},\mathsf{Bob},\mathsf{Eve} &\rightsquigarrow \mathsf{Alice},\mathsf{Bob} \\ \mathsf{Alice} &\rightsquigarrow \mathsf{Eve} \\ \mathsf{Bob} &\rightsquigarrow \mathsf{Eve} \end{aligned} \tag{3}$$



Π_1 Π_2 $\Pi_1 \cup \Pi_2$ Π_3 Π_4

Fig. 5. Security policies, graphically.

that allows Alice, Bob and Eve to influence Alice and Bob, Alice to influence Eve, and Bob to influence Eve. Intuitively, this can be used to specify that both Alice and Bob can talk individually to the intermediary Eve, which in turn can talk to Alice and Bob in coordination with them.

For a clause in $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ the superscript $\sigma \subseteq \mathscr{C}$ allows one to specify some additional coordination constraints on the interaction pattern among the levels in $\mathbb{A}$ and $\mathbb{B}$. The superscript d requires all entities in $\mathbb{A}$ to influence all the entities in $\mathbb{B}$ *directly*. For instance, for the policy (1) we might want Alice and Bob to influence Eve directly, with no delegation among them or to another intermediary. This leads to the policy $\Pi_1 \triangleq \mathsf{Alice}, \mathsf{Bob} \overset{d}{\rightsquigarrow} \mathsf{Eve}$ depicted in Figure 5.

In general the flow of information to all of the entities in $\mathbb{B}$ is just potential, i.e., the information is made available to entities in $\mathbb{B}$ simultaneously, but it could happen that after one entity in $\mathbb{B}$ acquired the information (some of) the others get disabled. The superscript f, that stands for "fair" disclosure, prevents the above to happen: whenever information flows to one of the entities in $\mathbb{B}$, then it must eventually flow to all other entities in $\mathbb{B}$.

It is worth to remark that the information flowing to the entities in $\mathbb{B}$ need not be the same since causality in our approach represents in general the transfer of *some* information. Ensuring that the *same* information is being transferred depends on the actual causal semantics under consideration.

In our previous examples, for policies (2) and (3) it is natural to require fairness to forbid competition among Alice and Bob. This leads to the policies Π_2 and Π_3 in Figure 5. Observe that fairness constraints are superfluous when there is only one security level in the target, like Eve in Π_1. Notice that in policy Π_3, the absence of the "direct" constraint allows Eve to act as an intermediary. A variant of Π_3 without intermediary is $\Pi_4 \triangleq \mathsf{Alice}, \mathsf{Bob} \overset{d,f}{\rightsquigarrow} \mathsf{Alice}, \mathsf{Bob}$, which specifies a direct exchange of information between Alice and Bob.

4.2 Semantics of Many-to-Many Policies

In our setting, a *security model* is an event structure used to capture the structure of computations and flows. Events correspond to actions of some security level.

Definition 6 (causal security model). *Let $\mathscr{L}$ be a set of security levels. A* (causal) security model *in $\mathscr{L}$ is a pair $\langle \mathcal{E}, \lambda \rangle$ where $\mathcal{E} = \langle E, \leq, \# \rangle$ is an event structure and $\lambda : E \to \mathscr{L}$ is a* security assignment.

The security assignment λ maps each event to a security level and can be lifted to sets of events: given $X \subseteq E$ we let $\lambda X = \{\mathsf{A} \in \mathscr{L} \mid \exists e \in X \,.\, \lambda(e) = \mathsf{A}\}$. For $X \subseteq E$ and $\mathbb{A} \subseteq \mathscr{L}$ we write $X : \mathbb{A}$ if $|X| = |\mathbb{A}|$ and $\lambda X = \mathbb{A}$. We write $e : \mathsf{A}$ instead of $\{e\} : \{\mathsf{A}\}$. The event structures of Figures 1–3a are security models, where the security assignment corresponds to the principal annotations.

We next formalise in which sense a clause $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ justifies a situation in which we have sets of events X and Y such that $X : \mathbb{A}$, $Y : \mathbb{B}$ and the events in X cause the events in Y. We do this by introducing a semantic counterpart of the relation $\overset{\sigma}{\rightsquigarrow}$ over sets of events. We first define some properties of sets of events that are related to the coordination constraints. We say that a set of events $X \subseteq E$ is *flat* if $[X)$ is a configuration and for every $e, e' \in X$ it holds $e \not\leq e'$. Notice that when X is flat, all events in X are are enabled at $[X)$. The events in a flat set X may disable each other or there can be future events that disable some of them but not the others. We say that X is *fair* if for all events $e, e' \in X$ and event $e'' \in E$, we have $e \# e''$ iff $e' \# e''$. Note that in this case, since the events in X have the same conflicts and conflict is irreflexive, either all or none of them is executable.

Definition 7 (relation $\overset{\sigma}{\twoheadrightarrow}$). *Let $\mathcal{E} = \langle E, \leq, \# \rangle$ be an event structure and let $X, Y \subseteq E$. We write*

(i) $X \twoheadrightarrow Y$ if $X < Y$, and Y is flat;
(ii) $X \overset{d}{\twoheadrightarrow} Y$ if $X \twoheadrightarrow Y$ and $X \lessdot Y$;
(iii) $X \overset{f}{\twoheadrightarrow} Y$ if $X \twoheadrightarrow Y$ and Y is fair;
(iv) $X \overset{d,f}{\twoheadrightarrow} Y$ if $X \overset{d}{\twoheadrightarrow} Y$ and $X \overset{f}{\twoheadrightarrow} Y$.

In words, we write $X \twoheadrightarrow Y$ whenever each event in X is a cause for each event in Y, and the set Y is flat, ensuring that $[Y)$ is a configuration enabling all events in Y.

We write $X \overset{d}{\twoheadrightarrow} Y$ when additionally $X \lessdot Y$, i.e., causality is direct, meaning that delegation is not permitted.

Notice that when $X \twoheadrightarrow Y$ events in Y are all enabled at configuration $[Y)$, so that the possibility of getting information from X is granted to all elements of Y, but once an event in Y is executed other events in Y could get disabled: it may be the case that the events in Y disable each other or, more generally, they could be in conflict with different events. The constraint $X \overset{f}{\twoheadrightarrow} Y$, asking that Y is fair, guarantees, instead, that if the information reaches one of the events in Y it will eventually reach all of them, or, more formally, that any configuration intersecting Y extends to a configuration including Y. Note that, in absence of conflicts, the fairness constraint becomes inessential since it is always trivially satisfied.

We can now define what it means for a security model to satisfy a policy.

	$\Pi_1 \cup \Pi_2$	Π_3	Π_4
P_1			
P_3			✓
P_2	✓	✓	
P_4			
P_5		✓	
P_6		✓	

Table 1. Models versus policies.

Definition 8 (security). *Let $\Pi = \langle \mathscr{L}, \rightsquigarrow \rangle$ be an information flow policy, $\mathcal{E} = \langle E, \leq, \# \rangle$ be an event structure, let $\langle \mathcal{E}, \lambda \rangle$ be a security model in $\mathscr{L}$. Given $X, Y \subseteq E$ and $\sigma \subseteq \mathscr{C}$, we say that the clause $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B} \in \Pi$* allows *$X \overset{\sigma}{\twoheadrightarrow} Y$, written $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B} \models X \overset{\sigma}{\twoheadrightarrow} Y$ if $X : \mathbb{A}$, $Y : \mathbb{B}$.*

We say that $\langle \mathcal{E}, \lambda \rangle$ satisfies Π or that $\langle \mathcal{E}, \lambda \rangle$ is Π-secure, denoted $\langle \mathcal{E}, \lambda \rangle \models \Pi$, if for all $e, e' \in E$ such that $e \lessdot e'$ there exist $X, Y \subseteq E$ with $e' \in Y$ such that $X \overset{\sigma}{\twoheadrightarrow} Y$ and $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B} \models X \overset{\sigma}{\twoheadrightarrow} Y$ for some $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B} \in \Pi$ such that $\lambda(e) \in \mathbb{A}$.

When λ is clear from the context we sometimes write $\mathcal{E} \models \Pi$.

Intuitively, a security model satisfies a policy if any direct causality $e \lessdot e'$ is part of a (possibly larger) interaction $X \overset{\sigma}{\twoheadrightarrow} Y$ justified by some clause $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ of the policy. We require $e' \in Y$ in order to ensure that the flow to e' is influenced by $X : \mathbb{A}$ (recall that $X < Y$) and thus by all levels in $\mathbb{A}$. On the other hand, event e is not necessarily in X since the information of level $\lambda(e)$ that is required to flow to all levels in $\mathbb{B}$ might have been provided by another cause of e'. Still, since e is a cause of e', it will be part of any computation involving e', hence it will coordinate with the events in X to enable e'. This is fundamental, e.g., to implement a simultaneous disclosure asynchronously as we shall explain later.

Table 1 summarizes the satisfaction of policies by the protocols of our running example. The double horizontal lines separate protocols without intermediaries (P_1,P_3) from those with intermediaries (P_2,P_4,P_5,P_6), and the vertical double lines separates policies with intermediaries ($\Pi_1 \cup \Pi_2$ and Π_3) from policies without intermediaries (Π_4). We start discussing the scenarios with intermediaries.

The security model of Figure 2b (Protocol P_2) satisfies the policies $\Pi_1 \cup \Pi_2$ and Π_3 of Figure 5. For example, the direct causality $\mathsf{put}\langle\mathsf{secA}\rangle@\mathsf{AE} : \mathsf{A} \lessdot \mathsf{gather}\langle?\mathsf{sec1}, ?\mathsf{sec2}\rangle@\mathsf{AE}, \mathsf{BE} : \mathsf{E}$ is justified by clause $\mathsf{Alice}, \mathsf{Bob} \overset{d}{\rightsquigarrow} \mathsf{Eve}$ of policy Π_1, by choosing $X = \{\mathsf{put}\langle\mathsf{secA}\rangle@\mathsf{AE} : \mathsf{A}, \mathsf{put}\langle\mathsf{secB}\rangle@\mathsf{BE} : \mathsf{B}\}$ and $Y = \{\mathsf{gather}\langle?\mathsf{sec1}, ?\mathsf{sec2}\rangle@\mathsf{AE}, \mathsf{BE} : \mathsf{E}\}$. In words, the disclosure to Eve is allowed since it is collectively made by Alice and Bob. The direct causalities between the multicast of Eve and the receptions by Alice and Bob are justified by the clauses in policies Π_2 or Π_3. In order to see this for the only clause of Π_2, observe that we can choose $X = \{\mathsf{scatter}\langle\mathsf{sec2}, \mathsf{sec1}\rangle@\mathsf{EA}, \mathsf{EB} : \mathsf{E}\}$ and $Y = \{\mathsf{get}\langle?\mathsf{sec}\rangle@\mathsf{EA} : \mathsf{A}, \mathsf{get}\langle?\mathsf{sec}\rangle@\mathsf{EB} : \mathsf{B}\}$. In words, both message receptions of Alice and Bob depend on Eve sending the secrets, and they cannot be disabled,

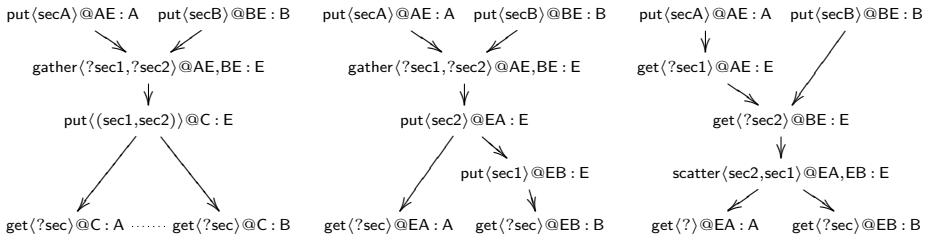


Fig. 6. Secret exchange protocols P_4 (left), P_5 (center) and P_6 (right).

hence they have the same (i.e., no) conflicts. For the only clause of Π_3 the situation is a bit more involved: the idea is to select Y as before and X to be the rest of the events. Intuitively, Eve acts as a delegated intermediary: the submission of the secrets by Alice and Bob influence their final receptions, indirectly through the events of Eve.

Consider now the event structures in Figure 6, which represent variants of the protocols of our running example where Eve is used as an intermediary. They can be seen as alternatives to the protocol P_2 in Figure 2b. Protocol P_4 is like P_2 but the intermediary does not scatter the secrets; these are instead combined in a composite message and sent to a channel C thus creating a conflict between Alice and Bob since both expect to extract the (combined) secrets from C. In protocol P_5, Eve sends the messages asynchronously with point-to-point operations, first to Alice and then to Bob. Finally, in P_6, the reception of the secrets is asynchronous, with point-to-point operations.

None of these variants satisfies policy $\Pi_1 \cup \Pi_2$. For instance, no clause in $\Pi_1 \cup \Pi_2$ justifies the direct causality $\mathsf{put}\langle(\mathsf{sec1},\mathsf{sec2})\rangle@\mathsf{C} : \mathsf{E} \lessdot \mathsf{get}\langle?\mathsf{sec}\rangle@\mathsf{C} : \mathsf{A}$ in P_4 due to the conflict between $\mathsf{get}\langle?\mathsf{sec}\rangle@\mathsf{C} : \mathsf{A}$ and $\mathsf{get}\langle?\mathsf{sec}\rangle@\mathsf{C} : \mathsf{B}$. In P_5 the direct causalities from Eve's put events to the get events of Alice and Bob cannot be justified since there is no Eve-labelled event that is a direct cause for both get events. Notice that such causalities could be justified if we drop the directness constraints from the clauses since the reception of the messages by Bob and Alice are causally dependent by the first put of Eve. Similarly, the direct causalities from Alice and Bob's put events to Eve's get events in P_6 cannot be justified.

The situation is different for Π_3. Indeed, both P_5 and P_6 satisfy the policy. Intuitively, the asynchronous collection of the secrets in P_6 that could not be justified in Π_2 can now be justified since Alice and Bob are allowed to talk to Eve independently. On the other hand, the asynchronous disclosure by Eve in P_5 is justified since it also depends on both Alice and Bob without directness constraint.

Similarly, it can be seen that among the protocols not using intermediaries, namely P_1 (Figure 2a) and P_3 (Figure 3a), only P_3 satisfies the policy Π_4. Protocol P_3 is indeed the only one that guarantees that Alice and Bob collectively make their secrets available to Alice and Bob simultaneously. Indeed, protocol

P_3 has a unique advantage over P_1: when Alice (resp. Bob) gets the secrets, (s)he is not ensured to be in competitive advantage. Clearly, protocol P_1 does not have this property. Therefore, P_3 offers a solution to the simultaneous secret exchange problem with guarantees based on causality rather than on bounds on the amount of different information obtained by the parties (as e.g. in [8]).

4.3 Semantic properties

We next present some properties of security policies. We first observe that aggregating security levels preserves the satisfaction of policies.

Proposition 1 (soundness of level aggregation). *Let $\langle \mathcal{E}, \lambda \rangle$ be a security model in $\mathscr{L}$ and $\rho : \mathscr{L} \to \mathscr{L}$, a total mapping between security levels (possibly merging some of them). If $\langle \mathcal{E}, \lambda \rangle \models \Pi$ then $\langle \mathcal{E}, \rho \circ \lambda \rangle \models \rho(\Pi)$.*

In the above definition $\rho(\Pi)$ is the homomorphic lifting of ρ to policies, i.e., $\rho(\Pi \cup \Pi') = \rho(\Pi) \cup \rho(\Pi')$, $\rho(\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}) = \rho(\mathbb{A}) \overset{\sigma}{\rightsquigarrow} \rho(\mathbb{B})$ and $\rho(\emptyset) = \emptyset$.

Secondly, we discuss how policies can be related according to their strictness.

Definition 9 (strictness relation). *The strictness relation $\sqsubseteq \subseteq \mathcal{P} \times \mathcal{P}$ is defined as the least transitive and reflexive relation among policies closed under*

$$\frac{\Pi \sqsubseteq \Pi' \quad \Pi'_1 \subseteq \Pi_1}{\Pi \cup \Pi_1 \sqsubseteq \Pi' \cup \Pi'_1}\ (\textsc{Ctx}) \qquad \frac{\sigma' \subseteq \sigma}{\mathbb{A} \overset{\sigma'}{\rightsquigarrow} \mathbb{B} \sqsubseteq \mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}}\ (\textsc{Constr})$$

$$\frac{\mathbb{A} = \mathbb{A}' \cup \mathbb{A}''}{\{\mathbb{A}' \overset{\sigma}{\rightsquigarrow} \mathbb{B}, \mathbb{A}'' \overset{\sigma}{\rightsquigarrow} \mathbb{B}\} \sqsubseteq \mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}}\ (\textsc{SplitL}) \qquad \frac{|\mathbb{B}| = 1}{\mathbb{A} \overset{\sigma \cup \{f\}}{\rightsquigarrow} \mathbb{B} \sqsubseteq \mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}}\ (\textsc{ConstrF})$$

$$\frac{\mathbb{B} = \mathbb{B}' \cup \mathbb{B}''}{\{\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}', \mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}''\} \sqsubseteq \mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}}\ (\textsc{SplitR}) \qquad \frac{|\mathbb{A}| = |\mathbb{B}| = 1}{\mathbb{A} \overset{\sigma \cup \{d\}}{\rightsquigarrow} \mathbb{B} \sqsubseteq \mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}}\ (\textsc{ConstrD})$$

The intuition is the following. Rule Ctx says that the strictness relation is preserved under context closure (i.e., if $\Pi \sqsubseteq \Pi'$ then $\Pi \cup \Pi'' \sqsubseteq \Pi' \cup \Pi''$). In addition, the relation is preserved even if the weaker policy Π is embedded in a larger context since the addition of clauses to a policy makes it more permissive. By rule SplitL if the source $\mathbb{A}$ of a clause is split as $\mathbb{A}' \cup \mathbb{A}''$ then the clause can be relaxed by replacing it with two clauses having the same coordination constraints, the same targets and $\mathbb{A}'$ and $\mathbb{A}''$, respectively, as sources. This weakens the policy since any direct causality from an event of level $\mathsf{A} \in \mathbb{A}$ to an event of level $\mathsf{B} \in \mathbb{B}$ that is justified by $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ can be justified by $\mathbb{A}' \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ or by $\mathbb{A}' \overset{\sigma}{\rightsquigarrow} \mathbb{B}$. Rule SplitR is analogous, but with the split performed on the target. Rule Constr says that a clause can be relaxed by removing coordination constraints in σ. The last two rules ConstrF and ConstrD capture the fact that some constraints are trivially satisfied when clauses have a special shape. More precisely, given a clause $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$ with $|\mathbb{B}| = 1$ the fairness constraint is vacuously satisfied and this motivates rule ConstrF. If, additionally, $|\mathbb{A}| = 1$ the same applies to the directness constraint and this leads to rule ConstrD.

When $\Pi \sqsubseteq \Pi'$ we say that Π is less restrictive than Π'. For instance, the policies Π_1 and Π_2 of our example are less restrictive than $\Pi_1 \cup \Pi_2$, and $\Pi_1 \cup \Pi_2$ and Π_3 are incomparable.

An interesting result is that the strictness relation $\sqsubseteq$ is sound and complete with respect to policy satisfaction.

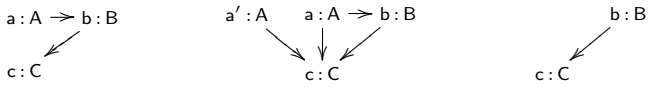
Proposition 2 ($\sqsubseteq$ is sound and complete). *Let $\Pi = \langle \mathscr{L}, \rightsquigarrow \rangle, \Pi' = \langle \mathscr{L}, \rightsquigarrow' \rangle$ be two policies. The following holds*

(i) if $\Pi' \sqsubseteq \Pi$ then, for all models $\langle \mathcal{E}, \lambda \rangle$ in $\mathscr{L}$, $\langle \mathcal{E}, \lambda \rangle \models \Pi$ implies $\langle \mathcal{E}, \lambda \rangle \models \Pi'$;
(ii) if for all models $\langle \mathcal{E}, \lambda \rangle$ in $\mathscr{L}$, $\langle \mathcal{E}, \lambda \rangle \models \Pi$ implies $\langle \mathcal{E}, \lambda \rangle \models \Pi'$, then $\Pi' \sqsubseteq \Pi$.

By soundness, whenever $\Pi \sqsubseteq \Pi' \sqsubseteq \Pi$, the policies Π and Π' are equivalent, i.e., they are satisfied by exactly the same models. Syntactically different policies can be equivalent because they have clauses that differ for constraints which are vacuosly satisfied (see rules ConstrF and ConstrD). Moreover, equivalent policies can differ for the presence of clauses which are useless because they are subsumed by others (e.g., if a policy includes the clauses $\mathbb{A} \overset{\mathscr{C}}{\rightsquigarrow} \mathbb{B}$, and $\mathbb{A}' \overset{\mathscr{C}}{\rightsquigarrow} \mathbb{B}$, the addition of a clause $\mathbb{A} \cup \mathbb{A}' \overset{\mathscr{C}}{\rightsquigarrow} \mathbb{B}$ produces an equivalent policy).

It can be proved that the partial order induced by the preorder $(\mathcal{P}, \sqsubseteq)$ is a complete lattice. The (equivalence class of the) policy $\{(\mathsf{A}, \mathscr{C}, \mathsf{A}) \mid \mathsf{A} \in \mathscr{L}\}$ is the top element (the most restrictive policy, which admits flows within individual security levels only), and (the equivalence class of) $\mathscr{L} \times \emptyset \times \mathscr{L}$ is the bottom element (the most permissive policy, which accepts all flows).

It is interesting to observe that the most restrictive policy satisfied by a security model does not exist in general. In fact, in order to determine such policy the idea could be to start with the most permissive policy, allowing for all binary flows, and keep using the rules (by joining or removing clauses, or adding coordination constraints) in order to restrict it until no more rules can be applied. This works, but the policy obtained is not uniquely determined, not even up to equivalence. For example, consider the security model $\mathcal{M}$ on the left of the figure below.



and the policies $\Pi_1 = (\mathsf{A}, \mathsf{B} \rightsquigarrow \mathsf{C}), (\mathsf{A} \rightsquigarrow \mathsf{B})$ and $\Pi_2 = (\mathsf{A} \rightsquigarrow \mathsf{B}), (\mathsf{B} \rightsquigarrow \mathsf{C})$. Clearly, $\mathcal{M}$ satisfies both Π_1 and Π_2. Such policies are incomparable with respect to $\sqsubseteq$ (none of them can be obtained from the other by removing or joining classes, and indeed event structures can be found that satisfy one of them but not the other like those at the middle and the right of the figure above). Hence Π_1 and Π_2 are distinct minimal policies satisfied by model $\mathcal{M}$: the most restrictive policy satisfied by $\mathcal{M}$ does not exist.

4.4 Decidability on Regular Trace Event Structures

The event structure associated with a system exhibiting a cyclic behaviour is infinite, since events represent "occurrences" of computational steps. Nevertheless, we can show that policy satisfaction is decidable once we focus on regular trace event structures [24], a class of "effectively represented" event structures enjoying suitable regularity properties. This result relies on the observation that policy satisfaction can be expressed as a first order property and first order logic (FOL) is known to be decidable on regular trace event structures [12].

Roughly speaking, for regular trace event structures dependencies are required to be induced by a finite labelling, endowed with an independence relation. More precisely, recall that a *trace alphabet* consists of a pair $M = \langle \Sigma, I \rangle$ where Σ is a finite label alphabet and $I \subseteq \Sigma \times \Sigma$ is an irreflexive relation called the independence relation. An M-labelled event structure is an event structure $\mathcal{E}$, with a labelling function $\xi : E \to \Sigma$ satisfying (1) if $e \#_\mu e'$ then $\xi(e) \neq \xi(e')$; (2) if $e \lessdot e'$ or $e \#_\mu e'$ then $(\xi(e), \xi(e')) \notin I$; and (3) if $(\xi(e), \xi(e')) \notin I$ then $e \leq e'$ or $e \# e$. Conditions (2) and (3) ensure that the concurrency relation of the event structure (unordered events that are not in conflict) conforms to the independence relation of M. Condition (1) asks that the Σ-labelled transition system associated with $\mathcal{E}$ is deterministic. In addition, as in [12], we require the regularity of the set of (labelled) sequences of execution from the empty configuration in the event structure, i.e., of the set $seq(\mathcal{E}) = \{\sigma \in \Sigma^* \mid \emptyset \xrightarrow{\sigma}{}^* C \text{ in } TS(\mathcal{E})\}$.

Definition 10 (regular trace event structure). *An event structure $\mathcal{E}$ is* regular trace *if it is M-labelled for some trace alphabet $M = \langle \Sigma, I \rangle$ and $seq(\mathcal{E}) \subseteq \Sigma^*$ is a regular language.*

When we need to make the trace alphabet explicit, we say that the event structure $\mathcal{E}$ is M-regular trace. The class of regular trace event structures is quite general. In [23] it has been shown to coincide with the class of event structures associated with finite safe Petri nets by the unfolding construction [16].

We first instantiate the notion of regularity in the setting of security models.

Definition 11 (regular trace model). *A security model $\langle \mathcal{E}, \lambda \rangle$ in $\mathscr{L}$ is* regular trace *if there exists a trace alphabet $M = \langle \Sigma, I \rangle$ such that $\mathcal{E}$ is M-regular trace via the labelling $\xi : E \to \Sigma$ and $\lambda = \lambda' \circ \xi$ for some $\lambda' : \Sigma \to \mathcal{L}$.*

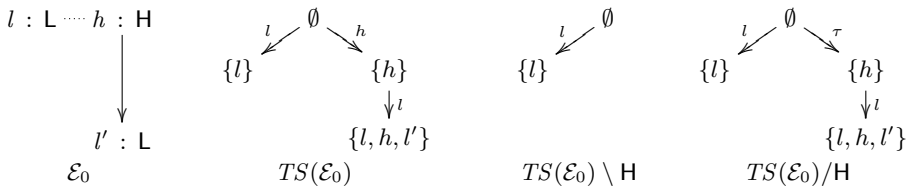
The encoding of policy satisfaction as a FOL sentence crucially relies on the fact that, in order to check whether a causal dependency $a \lessdot b$ is justified by some clause $\mathbb{A} \overset{\sigma}{\rightsquigarrow} \mathbb{B}$, we need to find sets of events $X : \mathbb{A}$, $Y : \mathbb{B}$ with bounded cardinality $|X| = |\mathbb{A}|$ and $|Y| = |\mathbb{B}|$. Then decidability follows from [12].

Proposition 3 (decidability on regular trace security models). *Let $\Pi = \langle \mathscr{L}, \rightsquigarrow \rangle$ be an information flow policy and let $\langle \mathcal{E}, \lambda \rangle$ be a regular trace security model in $\mathscr{L}$. The decision problem $\langle \mathcal{E}, \lambda \rangle \models \Pi$ is decidable.*

5 Non-Interference

We discuss in this section how our causality-based notion of information flow compares with notions of information flow based on non-interference for concurrent systems. The key differentiating factor resides in the semantic model: causal semantics are normally finer than interleaving semantics. The paradigmatic example is given by processes $a \mid b$ and $a.b + b.a$ which are equated in an interleaving world but are different from a causal point of view. Hence we would expect our notion of security, when confined to binary policies, to be more restrictive than those based on interleaving semantics. However, this is not always the case mainly because observational non-interference approaches capture flows of information due to conflicts (i.e., branching), which are instead not considered in our notion of security. We will also briefly discuss how our approach could be amended to consider conflict-related flows.

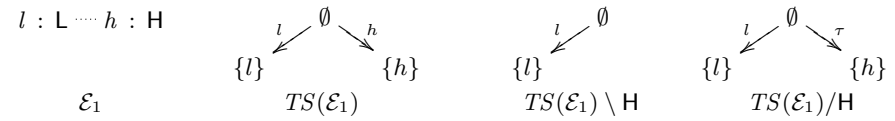
Consider, for example *Strong Non-deterministic Non-Interference* (SNNI) [6,7,3]. Informally, a system S is SNNI whenever $S \setminus \mathsf{H}$ (i.e., the low level system in isolation) and S/H (i.e., the entire system where high H actions are unobservable) are weak trace equivalent. Let us say that an event structure $\mathcal{E}$ is SNNI whenever its transition system $TS(\mathcal{E})$ is SNNI. Then it is easy to see that $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-security is strictly finer than SNNI. The easiest case is to show that SNNI does not imply $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-security. The counterexample is the event structure $\mathcal{E}_0$ below:



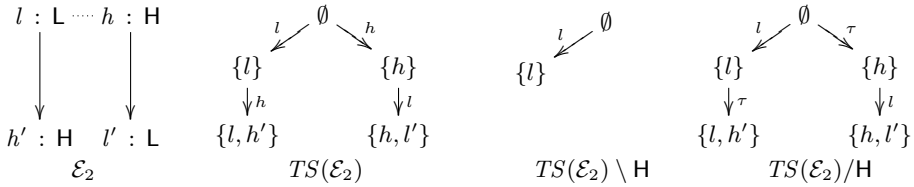
Clearly $\mathcal{E}_0$ is SNNI since the set of traces of both $\mathcal{E}_0 \setminus \mathsf{H}$ and $\mathcal{E}_0/\mathsf{H}$ is $\{\epsilon, l\}$, but $\mathcal{E}_0$ is not $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-secure since the direct causality $h \lessdot l'$ cannot be justified.

On the other hand, $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-security implies SNNI: if $\mathcal{E}_0$ is $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-secure then it contains no direct causality $e \lessdot e'$ with $e : \mathsf{L}$ and $e' : \mathsf{H}$, and hence all traces are such that high transitions are never necessarily followed by low transitions. Hence, the traces of $\mathcal{E}_0 \setminus \mathsf{H}$ and the traces of $\mathcal{E}_0/\mathsf{H}$ coincide.

Observational non-interference properties are expressive enough to capture information leakages arising from deadlocks and divergence. Particularly popular in this tradition are notions based on weak bisimulation, like the Bisimulation-based SNNI (BSNNI) and the finer Bisimulation-based Non-Deducibility on Compositions (BNDC) [6,7]. BSNNI is like SNNI but with weak bisimulation equivalence instead of trace equivalence. Note that BSNNI and $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-security are incomparable. First, it is easily seen that $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-security does not imply BSNNI.



The event structure $\mathcal{E}_1$ above is $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-secure (since there is no direct causality between l and h) but is not BSNNI, since $TS(\mathcal{E}_1)\setminus\mathsf{H}$ and $TS(\mathcal{E}_1)/\mathsf{H}$ are not weak bisimilar. Informally, the low part of the system can deduce that h occurred by observing that l is not executable. Vice versa, one can show that BSNNI does not imply $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-security. Consider the event structure $\mathcal{E}_2$ below.



Clearly, $\mathcal{E}_2$ is not $\{\mathsf{L} \rightsquigarrow \mathsf{H}\}$-secure since the direct causality $h \lessdot l'$ is not allowed. On the other hand, $\mathcal{E}_2$ is BNNI since $TS(\mathcal{E}_2)\setminus\mathsf{H}$ and $TS(\mathcal{E}_2)/\mathsf{H}$ are weak bisimilar.

For similar reasons, our notion of security is also incomparable to other observational notions of non-interference based on transition systems such as BNDC and those that extend [18] by requiring the source and the target states of every high transition to be (low) trace equivalent [1,7,9].

6 Related Work

Logic-based languages have been investigated as flexible formalisms to express security properties. We mention, e.g., *hyperproperties* [5] and their logics (e.g. [4,15]). Our policies can be easily integrated in logic-based languages to obtain a richer policy language. For instance, the simplest solution is to combine our policies by using standard propositional operators.

Another example is Paralocks [2], a language for specifying role-based information flow policies. In Paralocks a data item x can be annotated with clauses of the form $\Sigma \to a$, specifying the conditions Σ under which x can flow into actor a. Such policies can be related to many-to-one policies in our approach, i.e., of the form $\mathsf{A}_1 \ldots \mathsf{A}_m \rightsquigarrow \mathsf{B}_1$. It less clear how many-to-many policies, i.e., policies stating that data can flow to an agent a if it also flows to an agent b, could be expressed in Paralocks.

In addition to FOL and MSO for event structures [12], our work can be related to other logics motivated by causality. An example is *Event Order Logic* (EOL) of [10], a logic based language used to specify boolean conditions on the occurrence of events, with applications to safety analysis. EOL is inspired by Linear-time Temporal Logic (LTL) and introduces ad-hoc operators to specify the order of events. The more relevant operator is $\psi_1 \wedge \psi_2$ which allows one

to express that the events described by ψ_1 occur before those described by ψ_2. Hence, a policy $\mathsf{A}_1 \dots \mathsf{A}_m \rightsquigarrow \mathsf{B}_1 \dots \mathsf{B}_n$ can be related to an EOL formula $\mathsf{A}_1 \vee \dots \vee \mathsf{A}_m \wedge \mathsf{B}_1 \vee \dots \vee \mathsf{B}_n$. However, EOL does not feature operators to express some of our coordination constraints.

7 Conclusion

We have presented a novel approach to many-to-many information flow policies that allows one to specify patterns of coordination among several security levels. In particular, each clause in a policy can specify that a certain set of levels is allowed to cooperate to provide the flow *collectively*, and possibly *simultaneously* to another set of security levels. We believe that the approach can turn useful in several scenarios including secret exchange protocols, security architectures and systems with controlled forms of declassification. We have provided decidability results and discussed the relation to some traditional notions of security, including observational non-interference and language-based security. We are currently investigating the development of verification and enforcement mechanisms for concrete specification languages, like programming languages and Petri nets. We are also investigating extensions of our work in several directions, including how to suitably deal with indirect flows of information due to conflicts, and how to deal with the transfer of specific values.

References

1. A. Bossi, C. Piazza, and S. Rossi. Modelling downgrading in information flow security. In *Proceedings of CSFW 2004*, page 187. IEEE Computer Society, 2004.
2. N. Broberg and D. Sands. Paralocks: role-based information flow control and beyond. In M. V. Hermenegildo and J. Palsberg, editors, *Proceedings of POPL 2010*, pages 431–444. ACM, 2010.
3. N. Busi and R. Gorrieri. Structural non-interference in elementary and trace nets. *Mathematical Structures in Computer Science*, 19(6):1065–1090, 2009.
4. M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In M. Abadi and S. Kremer, editors, *Proceedings of POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.
5. M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
6. R. Focardi and R. Gorrieri. A taxonomy of security properties for process algebras. *Journal of Computer Security*, 3(1):5–34, 1995.
7. R. Focardi and R. Gorrieri. Classification of security properties (part I: information flow). In R. Focardi and R. Gorrieri, editors, *Proceedings of FOSAD 2000*, volume 2171 of *LNCS*, pages 331–396. Springer, 2000.
8. E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In H. Imai and Y. Zheng, editors, *Proceedings of PKC 1999*, volume 1560 of *LNCS*, pages 53–68. Springer, 1999.

9. R. Gorrieri and M. Vernali. On intransitive non-interference in some models of concurrency. In A. Aldini and R. Gorrieri, editors, *Proceedings of FOSAD 2010*, volume 6858 of *LNCS*, pages 125–151. Springer, 2011.
10. F. Leitner-Fischer and S. Leue. Probabilistic fault tree synthesis using causality computation. *IJCCBS*, 4(2):119–143, 2013.
11. X. Li, F. Nielson, H. R. Nielson, and X. Feng. Disjunctive information flow for communicating processes. In P. Ganty and M. Loreti, editors, *Proceedings of TGC 2015*, volume 9533 of *LNCS*, pages 95–111. Springer, 2015.
12. P. Madhusudan. Model-checking trace event structures. In *Proceedings of LICS 2013*, pages 371–380. IEEE Computer Society, 2003.
13. H. Mantel and D. Sands. Controlled declassification based on intransitive noninterference. In W. Chin, editor, *Proceedings of APLAS 2004*, volume 3302 of *LNCS*, pages 129–145. Springer, 2004.
14. R. C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978.
15. D. Milushev and D. Clarke. Towards incrementalization of holistic hyperproperties. In P. Degano and J. D. Guttman, editors, *Proceedings of POST 2012*, volume 7215 of *LNCS*, pages 329–348. Springer, 2012.
16. M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains. In G. Kahn, editor, *Proceedings of the International Symposium on the Semantics of Concurrent Computation*, volume 70 of *LNCS*, pages 266–284. Springer, 1979.
17. T. Okamoto and K. Ohta. How to simultaneously exchange secrets by general assumptions. In D. E. Denning, R. Pyle, R. Ganesan, and R. S. Sandhu, editors, *CCS '94, Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 184–192. ACM, 1994.
18. J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, Stanford Research Institute, 1992.
19. J. Rushby. Separation and integration in MILS (the MILS constitution). Technical report, Computer Science Laboratory SRI International, 2008.
20. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
21. A. Sabelfeld and D. Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.
22. Selective disclosure and insider trading. Technical report, U.S. Securities and Exchange Commission (SEC), August 2000.
23. P. Thiagarajan. Regular trace event structures. Technical Report RS-96-32, BRICS, 1996.
24. P. S. Thiagarajan. Regular event structures and finite Petri nets: A conjecture. In *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 244–256. Springer, 2002.
25. G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course*, volume 255 of *LNCS*, pages 325–392. Springer, 1986.