A Logic for Information Flow Analysis of Distributed Programs

Musard Balliu

School of Computer Science and Communication
KTH Royal Institute of Technology, Stockholm, Sweden
musard@kth.se

Abstract. Securing communication in large scale distributed systems is an open problem. When multiple principals exchange sensitive information over a network, security and privacy issues arise immediately. For instance, in an online auction system we may want to ensure that no bidder knows the bids of any other bidder before the auction is closed. Such systems are typically interactive/reactive and communication is mostly asynchronous, lossy or unordered. Language-based security provides language mechanisms for enforcing end-to-end security. However, with few exceptions, previous research has mainly focused on relational or synchronous models, which are generally not suitable for distributed systems.
This paper proposes a general knowledge-based account of possibilistic security from a language perspective and shows how existing trace-based conditions fit in. A syntactic characterization of these conditions, given by an epistemic temporal logic, shows that existing model checking tools can be used to enforce security.

Keywords: possibilistic information flow, logic of knowledge, language-based security, verification

1 Introduction

The emergence of ubiquitous computing paradigm makes software security more and more a real concern. Web browsers, smartphones, clouds are only few examples where untrusted and partially trusted code is regularly executed alongside applications processing personal sensitive data. In addition, current trends in computing such as code mobility and platform independence make the situation even worse. Attackers can then exploit vulnerabilities and deduce information about sensitive data by observing the behavior of malicious, or simply buggy, programs.

Information flow security policies [1], if successfully enforced or verified, prevent different types of confidentiality and integrity attacks. Language-based security provides end-to-end guarantees by means of programming language techniques. However, most work on language-based security models of information flow assumes synchronous or relational communication [2, 3, 4]. Although these

models are important in many settings, they are not obviously well suited for distributed programs where communication is interactive/reactive, nondeterministic and mostly asynchronous, lossy or unordered. The result is that programs that are considered insecure in one model may be secure in another, and vice versa.

Moreover, information flow policies are hard to verify in practice. The majority of static analyses for information flow security use standard methods such as security type systems [5, 3]. These analyses are efficient and attempt to ensure a strict separation, up to declassification and endorsement, between the sensitive part of the computation and the observable part of the computation. Obviously, if both parts are separated, it is impossible to learn anything about the sensitive data by observing the public data. Despite their efficiency, these methods lack the precision needed to handle programs where public and sensitive information are securely interwoven. Few works [6, 7, 8], at least in the setting of software security, attempt to deduce what is learned by observing the public effects of the computation, and then verify that the acquired knowledge does not break a given information flow policy toward sensitive data.

Motivating Example An online auction is a distributed system consisting of an auctioneer A and several bidders B_i competing for items I_k. Such systems are complex and usually involve both message passing and shared memory. For example, the auctioneer may receive messages from bidders who want to participate in the auction and associate a dedicated thread to each request. Then, depending on the auction protocol, each thread may read and write to a private shared array containing bids for all bidders and items. Several information flow policies may be worth enforcing in this scenario[1].

P_1 : The authentication code (pwd) of bidder B_i is always (G) secret wrt. any bidder B_j. In logic: $G\neg K_{B_j}(pwd_{B_i} = v)$.

P_2 : The sequence of bids of bidder B_i remains secret wrt. all bidders B_j until (W) the auction is closed. In logic: $L_{B_j}(secArray = v)\ W\ aClosed$.

P_3 : Only the first 3 bids of bidder B_i are considered secret wrt. any bidder B_j until the auction is closed. In logic: $L_{B_j}(\phi(b_1^i, b_2^i, b_3^i))\ W\ aClosed$.

P_4 : Any bid of bidder B_3 remains secret wrt. a colluding attack of B_1 and B_2. In logic: $GL_{B_1,B_2}(b^3 = v)$.

P_5 : The system may nondeterministically select a subset of bids from the private array, compute the maximum and promote an item I^* to the winner B^*. The output of this process may be considered secret wrt. any bidder $B_j \neq B^*$. In logic: $G\neg K_{B_j}(out(B^*, I^*))$.

As illustrated above, several issues should be handled to enforce the security policies of such systems. First, they are inherently nondeterministic, hence possibilistic notions of information flow security are needed. Second, distributed programs are usually interactive/reactive, which requires protection of sequences of (input or output) events as opposed to classical relational models where the

[1] The reader can already get the flavor of the logic used for security specifications.

input is read in the beginning of execution. Third, security policies are usually dynamic and involve controlled release of secret information. Finally, in distributed settings attackers may collude and share their observations to disclose secret information.

In this paper we model distributed systems in a trace-based setting where an execution trace is a sequence of events on channels. Security properties are expressed in terms of knowledge-based (epistemic) conditions over system traces. The security model brings out what events $\mathcal{O}$ on channels an observer can see and what observations on events $\mathcal{P}$ should be protected. Then the system is secure if the knowledge about events in $\mathcal{P}$ of an observer who makes observations in $\mathcal{O}$, at any point in the execution trace, is in accordance with the security policy at that point. Namely, the observer is unable to learn more information than what is allowed at a given point while moving to a successive point of the same trace and possibly making a new observation. This model fits well with current knowledge-based approaches to information flow security [9, 6, 10], and, inspired by work of Guttman and Nadel [11], by being explicit about the information that needs to be protected, it allows a very general treatment of secret information, both as high level input and output events, and as relationships between events, say ordering, multiplicity, and interleaving. We show that several possibilistic conditions such as Separability, Generalized Noninterference, Nondeducibility, Nondeducibility on Outputs and Nondeducibility on Strategies are accurately reflected in the epistemic setting.

Then we turn to the verification problem and present a linear time epistemic logic, with past time operators, which allows us to syntactically characterize security properties. The logic can be used as specification language for expressing possibilistic information flow policies. This enables modeling of the intricate and precise policies described in the motivating example and, at the same time, ensures separation between the actual code and the policy. Recent advances software model checking and automated theorem proving show that verification of temporal epistemic properties for distributed systems is feasible [12]. Our tool, ENCoVer [13], an extension of Java Pathfinder, can verify information flow policies for interactive sequential programs. However, scalability and complexity of verification are issues that we postpone to future work. An extended version of the present paper, which includes the proofs, can be found in [14].

2 Security Model

Program Model A model $\mathcal{M}$ is a set of finite or infinite traces induced by the program semantics. A trace τ is a sequence of actions relevant to the analysis. For instance, it can be messages sent over channels, read/write operations to shared memory, logical time ticks and so on. We write $|\tau|$ for the length (number of actions) of the trace τ. Whenever $|\tau| = \infty$, the trace has infinite length. A *point* is a pair (τ, i), where τ is a trace and $0 \leq i \leq |\tau|$. The function *trace* maps trace points to the prefix of the trace up to that point, namely $trace(\tau, i)$ denotes the sequence of actions α_j, where $0 \leq j < i$. In our setting, the actions

belong to a set $Act = \{\mathbf{out}(c, v), \mathbf{in}(c, v) \mid v \in Val, c \in Chan\} \cup \{\epsilon\}$, where Val is the domain of values and $\mathbf{in}(c, v)$ (resp. $\mathbf{out}(c, v)$) denotes the input (output) of value v on channel $c \in Chan$. The silent action is ϵ. We write $\tau_1 \bullet \tau_2$ for concatenation of two traces and $\tau \bullet \alpha$ for concatenation of trace τ with action α. The empty trace is ε and trace interleaving $\ltimes(\tau_1, \tau_2)$ is a set of traces coinductively defined as expected. The set inclusion is denoted as $\preceq$ and $\tau(i)$ is the i-th action of trace τ. The projection of a trace τ on a set of actions $\mathcal{A} \subseteq Act$ is defined as the subsequence of actions from $\mathcal{A}$ and denoted as $\tau_{\downarrow\mathcal{A}}$. Models can be enriched with structure by defining particular relations. In particular, given a poset $(S, \sqsubseteq)$, an upper closure operator (for short *uco*) is a function $\rho : S \to S$ such that $(a)\forall s \in S.\ s \sqsubseteq \rho(s)$, $(b)\forall s_1, s_2 \in S.\ s_1 \sqsubseteq s_2 \Rightarrow \rho(s_1) \sqsubseteq \rho(s_2)$, $(c)\forall s \in S.\ \rho(s) = \rho(\rho(s))$.

Security Policy We are mainly concerned with protecting confidentiality of actions on channels, hence we assume a set of security levels $\mathcal{L}$ for confidentiality and a relation $\sqsubseteq$ over $\mathcal{L}$. Moreover, we consider two observers (potentially sets of agents), one of security level $\mathtt{H}$ and the other of security level $\mathtt{L}$, which interact with the system by providing inputs and receiving outputs on channels of the same security level. Each agent has different clearance represented by a poset $(\mathcal{L}, \sqsubseteq)$ of two elements $\mathcal{L} = \{\mathtt{H}, \mathtt{L}\}$ with $\mathtt{L} \sqsubseteq \mathtt{H}$. A partial function $\mathcal{S} : Chan \rightharpoonup \mathcal{L}$, mapping channels to security levels, determines the set of channels accessible to each observer. Then the security policy is defined as a pair $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ where $\mathcal{O}$ is the set of channels that an observer can control and $\mathcal{P}$ is the set of channels to be protected. Usually we define $\mathcal{O} = \{c \in Chan \mid \mathcal{S}(c) = \mathtt{L}\}$ and $\mathcal{P} = \{c \in Chan \mid \mathcal{S}(c) = \mathtt{H}\}$. The fact that $\mathcal{S}$ is partially defined allows us to model channels which are invisible to the observer, yet not subject to protection.

Security Condition The security condition determines when a model $\mathcal{M}$ is secure with respect to a security policy $(\mathcal{O}, \mathcal{P})$. Here we define security in terms of the knowledge of an observer who knows the system specification[2] and interacts through channels in $\mathcal{O}$. The security condition prevents the observer from learning information about (properties of) interactions through channels in $\mathcal{P}$. First we define the observer knowledge at point (τ, i) as

$$\mathcal{K}(\tau, i, \mathcal{O}) = \{\tau' \mid \tau' \in \mathcal{M} \wedge (\tau', i') =_{\downarrow\mathcal{O}} (\tau, i) \wedge |i - i'| \leq t\}$$

where t is a *synchrony* parameter of the observer, $0 \leq i' \leq |\tau|$ and $(\tau', i') =_{\downarrow\mathcal{O}} (\tau, i)$ if the projection of (τ, i) and (τ', i') on actions in $\mathcal{O}$ is the same, namely, $(\tau', i')_{\downarrow\mathcal{O}} = (\tau, i)_{\downarrow\mathcal{O}}$. Intuitively, $\mathcal{K}(\tau, i, \mathcal{O})$ represents the set of traces that the observer considers possible based on its observations up to point (τ, i) and having synchrony parameter t. In particular, in a synchronous system $t = 0$, i.e. the observer knows the exact logical time. An asynchronous system can similarly be modeled by $t = \infty$. Models of semi-synchronous systems, where the observer

[2] In a language-based security setting the attacker is usually assumed to have complete knowledge of the program code.

knows the time approximately, can also be expressed. Then we define projection of trace τ on a set of channels $\mathcal{C}$, where $e(c, v)$ denotes an event on channel c.

$$\tau_{\downarrow\mathcal{C}} ::= \begin{cases} \varepsilon & \text{if } \tau = \varepsilon \\ e(c,v) :: \tau'_{\downarrow\mathcal{C}} & \text{if } c \in \mathcal{C} \text{ and } \tau = e(c,v) \bullet \tau' \\ \tau'_{\downarrow\mathcal{C}} & \text{if } c \notin \mathcal{C} \text{ and } \tau = e(c,v) \bullet \tau' \end{cases}$$

Notice that we leave the meaning of the :: operator undefined. By default we interpret :: as concatenation, however it need not be, as we will see when discussing different security policies.

At this point we have all ingredients to present the knowledge-based security condition. The intuition is simple: given a model $\mathcal{M}$ and a security policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$, the condition ensures that for each point $i+1$ of trace τ, the observer's knowledge about actions in $\mathcal{P}$ is not greater than its knowledge at the previous point i.

Definition 1 (Knowledge-based Security). *Let $\mathcal{M}$ be a model and $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ a security policy. Then $\mathcal{M}$ is* secure *wrt. $\mathcal{P}ol$ if for all $\tau \in \mathcal{M}$, $0 \leq i < |\tau|$*

$$\mathcal{K}(\tau, i, \mathcal{O})_{\downarrow\mathcal{P}} \preceq \mathcal{K}(\tau, i+1, \mathcal{O})_{\downarrow\mathcal{P}}$$

We illustrate the main idea behind the security condition with an example.

Example 1. Consider a program $P ::= \mathbf{in}(c_1, x); \mathbf{out}(c_2, x)$ which receives a boolean value on input channel c_1 and sends it on output channel c_2. The model $\mathcal{M}_P$ of P consists of two traces $\tau_1 = \mathbf{in}(c_1, \mathit{true}) \bullet \mathbf{out}(c_2, \mathit{true})$ and $\tau_2 = \mathbf{in}(c_1, \mathit{false}) \bullet \mathbf{out}(c_2, \mathit{false})$. The goal is to check whether $\mathcal{M}_P$ satisfies the policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$, where $\mathcal{O} = \{c_2\}$ and $\mathcal{P} = \{c_1\}$. Namely, we check if agent $\mathtt{L}$, the attacker, who knows $\mathcal{M}_P$ and observes values on channel c_2, can deduce information about activity of agent $\mathtt{H}$ on channel c_1. We identify agent $\mathtt{L}$ with $\mathcal{O}$ and agent $\mathtt{H}$ with $\mathcal{P}$. Then, applying Def. 2, we obtain ($k \in \{1, 2\}$):

- $\mathcal{K}(\tau_k, 0, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_k, 1, \mathtt{L})_{\downarrow\mathtt{H}} = \{\mathbf{in}(c_1, \mathit{true}), \mathbf{in}(c_1, \mathit{false})\}$
- $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}} = \{\mathbf{in}(c_1, \mathit{true})\}$ and $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow\mathtt{H}} = \{\mathbf{in}(c_1, \mathit{false})\}$

The program is insecure since $\mathcal{K}(\tau_1, 1, \mathtt{L})_{\downarrow\mathtt{H}} \not\preceq \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}}$. Namely, when the attacker observes $\mathbf{out}(c_2, \mathit{true})$, he refines his knowledge about secret actions from $\{\mathbf{in}(c_1, \mathit{true}), \mathbf{in}(c_1, \mathit{false})\}$ to $\{\mathbf{in}(c_1, \mathit{true})\}$ and deduces that value *true* was input on channel c_1 by agent $\mathtt{H}$.

The security condition in Def. 1 can be relaxed to deal with different forms of dynamic policies [15, 10]. A release (or declassification) policy $\mathcal{R}(\tau, i, \mathcal{P})$ at point (τ, i) is a property of $\mathcal{P}$, i.e., subset of $\mathcal{M}_{\downarrow\mathcal{P}}$, representing the knowledge that the observer is allowed to learn at that point. Consequently, a program is secure if the attacker's knowledge and the released knowledge at point (τ, i) is not greater than the attacker's knowledge at point $(\tau, i+1)$.

Definition 2 (Security wrt. Release). *Let $\mathcal{M}$ be a model with security policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ and release policy $\mathcal{R}(\tau, i, \mathcal{P})$. Then $\mathcal{M}$ is* secure *wrt. $\mathcal{P}ol$ and $\mathcal{R}(\tau, i, \mathcal{P})$ if for all $\tau \in \mathcal{M}$, $0 \leq i < |\tau|$*

$$\mathcal{K}(\tau, i, \mathcal{O})_{\downarrow\mathcal{P}} \cap \mathcal{R}(\tau, i, \mathcal{P}) \preceq \mathcal{K}(\tau, i+1, \mathcal{O})_{\downarrow\mathcal{P}}$$

If no action from $\mathcal{P}$ is released, $\mathcal{R}(\tau, i, \mathcal{P}) = \mathcal{M}_{\downarrow\mathcal{P}}$ and Def. 1 and Def. 2 coincide. Reconsider Ex. 1 with release policy $\mathcal{R}(\tau_1, 1, \mathtt{H}) = \{\mathbf{in}(c_1, \mathit{true})\}$ and $\mathcal{R}(\tau_2, 1, \mathtt{H}) = \{\mathbf{in}(c_1, \mathit{false})\}$. Then P is secure as $\mathcal{K}(\tau_1, 1, \mathtt{L})_{\downarrow\mathtt{H}} \cap \{\mathbf{in}(c_1, \mathit{true})\} \preceq \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}}$ and $\mathcal{K}(\tau_2, 1, \mathtt{L})_{\downarrow\mathtt{H}} \cap \{\mathbf{in}(c_1, \mathit{false})\} \preceq \mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow\mathtt{H}}$.

In a distributed setting, different agents may form coalitions and share observations in order to disclose secret information about other agents. The following definition gives a security condition in presence of colluding attacks.

Definition 3 (Security wrt. Collusion). *Let $\mathcal{M}$ be a model and two agents a_1, a_2 observing, resp., $\mathcal{O}_1, \mathcal{O}_2$. Then $\mathcal{M}$ secure wrt. a* colluding attack *on $\mathcal{P}$ if $\mathcal{M}$ is secure wrt. policy $(\mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{P})$.*

Example 2. Let P be a program with $c_1 \in \mathcal{P}$, $c_2 \in \mathcal{O}_1$ and $c_3 \in \mathcal{O}_2$. P is secure wrt. policies $\mathcal{P}ol_1 = (\mathcal{O}_1, \mathcal{P})$ and $\mathcal{P}ol_2 = (\mathcal{O}_2, \mathcal{P})$, but insecure wrt. a colluding attack, i.e., the policy $\mathcal{P}ol = (\mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{P})$.

$$P ::= \begin{bmatrix} \mathbf{in}(c_1, x) \\ \mathbf{if}\ x\ \mathbf{then}\ \mathbf{out}(c_2, 0) || \mathbf{out}(c_3, 1) \\ \mathbf{else}\ \mathbf{out}(c_2, 0); \mathbf{out}(c_3, 1) \end{bmatrix}$$

To see this, consider the program model $\mathcal{M}_P = \{\mathbf{in}(c_1, 1) \bullet \mathbf{out}(c_2, 0) \bullet \mathbf{out}(c_3, 1),$ $\mathbf{in}(c_1, 1) \bullet \mathbf{out}(c_3, 1) \bullet \mathbf{out}(c_2, 0),$ $\mathbf{in}(c_1, 0) \bullet \mathbf{out}(c_2, 0) \bullet \mathbf{out}(c_3, 1)\}$ where the secret on c_1 is binary and $||$ denotes the nondeterministic choice. If an agent merely observes the value received on his channel, there is nothing he can tell about the secret bit on c_1. However, if they collude, the observation of low sequence $\mathbf{out}(c_3, 1) \bullet \mathbf{out}(c_2, 0)$ reveals that the secret bit was 1.

Trace-based Conditions We next introduce several possibilistic information flow conditions from the literature and briefly discuss the flavor of each. The reason is two-fold; first to identify which aspects of security they enforce and, second, to show how these aspects can be captured by the knowledge-based conditions. We denote projection of trace τ on a set $\mathcal{A}$ as $\tau_{\mathcal{A}}$ (instead of $\tau_{\downarrow\mathcal{A}}$) to distinguish from the knowledge-based condition.

Separability was first introduced by McLean [16]. The goal is to ensure a logical separation between secret and public computations in both directions.

Definition 4 (Sep). *A model $\mathcal{M}$ satisfies* separability *if*

$$\forall \tau, \tau' \in \mathcal{M}, \forall \tau^* \in \ltimes(\tau_{\mathtt{L}}, \tau'_{\mathtt{H}}), \tau^* \in \mathcal{M}$$

A version of separability for synchronous systems has been proposed in [8].

Definition 5 (SSep). *A model $\mathcal{M}$ satisfies* synchronous separability *if*

$$\forall \tau, \tau' \in \mathcal{M},\ \exists \tau^* \in \mathcal{M}.\ \tau^*_{\mathtt{L}} = \tau_{\mathtt{L}} \text{ and } \tau^*_{\mathtt{H}} = \tau'_{\mathtt{H}}$$

Generalized noninterference is a relaxation of separability and it ensures that low computation is independent of the sequence of high inputs `HI` [16].

Definition 6 (GNI). *Model $\mathcal{M}$ satisfies* generalized noninterference *if*

$$\forall \tau, \tau' \in \mathcal{M}, \forall \tau^* \in \ltimes(\tau_{\mathtt{L}}, \tau'_{\mathtt{HI}}),\ \exists \tau'' \in \mathcal{M},\ \tau^* = \tau''_{\mathtt{L} \cup \mathtt{HI}}$$

Moreover, GNI prevents the low user from deducing information about both occurrences and non occurrences of high inputs. Nondeducibility, introduced by Sutherland [17], considers the system as a set of possible worlds W and defines security in terms of (information) functions f and g, such that for all $w_1, w_2 \in W$, there exists $w_3 \in W$ and $f(w_1) = f(w_3)$ and $g(w_2) = g(w_3)$. If one interprets f as computing the sequence of high input events and g as computing the sequence of low events, then nondeducibility can be defined as follows.

Definition 7 (ND). *A model $\mathcal{M}$ satisfies* nondeducibility *if*

$$\forall \tau, \tau' \in \mathcal{M},\ \exists \tau^* \in \mathcal{M}.\ \tau^*_{\mathtt{HI}} = \tau_{\mathtt{HI}} \text{ and } \tau^*_{\mathtt{L}} = \tau'_{\mathtt{L}}$$

One drawback of GNI and ND is that they are not adequate for systems that need to protect high output events or generate secrets internally. To solve this issue Guttman and Nadel introduced nondeducibility on outputs which prevents deductions of high events [11] and allows information flowing from low user inputs, here `LI`, to high outputs.

Definition 8 (NDO). *A model $\mathcal{M}$ satisfies* nondeducibility on outputs *if*

$$\forall \tau, \tau' \in \mathcal{M},\ \tau_{\mathtt{LI}} = \tau'_{\mathtt{LI}},\ \exists \tau^* \in \mathcal{M}.\ \tau^*_{\mathtt{H} \cup \mathtt{LI}} = \tau_{\mathtt{H} \cup \mathtt{LI}} \text{ and } \tau^*_{\mathtt{L}} = \tau'_{\mathtt{L}}$$

On the other hand, ND and GNI are too weak to ensure security for systems that exploit internal nondeterminism to transmit secrets through strategies implemented by high users [18]. A strategy is a function from sequences of high inputs and high outputs to values in a domain. A high user can use a strategy to compute the next input value on a high channel, as a function of the history of high values, and transmit information to a low user.

Definition 9 (NDS). *Let $\mathcal{M}$ be a model and $s_1, s_2 : \mathcal{M} \to Val$ two high strategies. Then $\mathcal{M}$ satisfies* nondeducibility on strategies *if*

$$\forall \tau, \tau' \in \mathcal{M},\ s_1(\tau_{\mathtt{L}}) = s_2(\tau'_{\mathtt{L}}) \Rightarrow \tau_{\mathtt{L}} = \tau'_{\mathtt{L}}$$

Most of the trace-based conditions assume either synchronous or asynchronous models. However, in language-based security, the knowledge of program code can give partial information about the order of events on high and low channels,and yet the program can be considered secure. We illustrate this fact with an example.

Example 3. Consider program P where $\{c_1, c_2\} \in \mathcal{P}$ and $\{c_3\} \in \mathcal{O}$.

$$\mathbf{in}(c_1, \mathit{secret}); \mathbf{out}(c_2, \mathit{secret}); \mathbf{out}(c_3, \text{"Done"})$$

The program receives a secret input from a high agent, writes to a file of the same agent and notifies the low agent that the operation is completed. In an asynchronous model, the secret is first received on c_1 and it is sent on c_2 or c_3 in any order. Hence $\mathcal{M}$ consists of the following traces:

$$\mathbf{in}(c_1, v_i) \bullet \mathbf{out}(c_2, v_i) \bullet \mathbf{out}(c_3, \text{"Done"}),$$
$$\mathbf{in}(c_1, v_i) \bullet \mathbf{out}(c_3, \text{"Done"}) \bullet \mathbf{out}(c_2, v_i)$$

It can be easily checked that $\mathcal{M}$ does not satisfy Sep and GNI, since the system is not closed under interleavings between H and L events. On the other hand, it seems reasonable to accept P as secure. The knowledge-based condition in Def. 1 accepts $\mathcal{M}$ wrt. policy $(\mathcal{O}, \mathcal{P})$ as do ND and NDO.

This example raises the question of what security policy a condition is enforcing and how these conditions can be interpreted in a unified framework.

3 Policies via Examples

We now introduce, by means of examples, different security policies using the epistemic security conditions. The set of channels in $\mathcal{P}$ and the set of channels in $\mathcal{O}$ are, resp., identified with H and L. Moreover, we redefine the semantics of the :: operator to handle different policies. We always define :: as concatenation when projecting on channels in $\mathcal{O}$. This reflects the assumption of perfect recall attacker with unbounded memory. Furthermore, given the set of high channels in $\mathcal{P}$, we write $Set(\mathtt{H})$ or just H to define :: as set union and $Mul(\mathtt{H})$ to define :: as multiset union. Finally, $Seq(\mathtt{H})$ defines :: as concatenation, while $Seq(\mathtt{H}\ \bot)$ defines :: as concatenation and replaces events in L with the special symbol $\bot$. For example, $(\mathtt{L}, Mul(\mathtt{H}))$ defines a policy which protects the multiplicity of high actions wrt. an attacker that observes actions in $\mathcal{O}$. Likewise, the policy $(\mathtt{L}, Seq(\mathtt{H}\ \bot))$ prevents the attacker from deducing information about interleavings of high actions in $\mathcal{P}$ with low actions in $\mathcal{O}$. Whenever the action type is unimportant, we write $l_1, l_2, \cdots$ for actions in L and $h_1, h_2, \cdots$ for actions in H. In the examples, traces are numbered as $\tau_1, \tau_2, \cdots$ following the order they appear in $\mathcal{M}$.

The first point we want to make is what happens in relational models of information flow where inputs are read in the beginning of program execution. All direct and implicit flows from high channels to low channels are captured by the security policy $(\mathtt{L}, \mathtt{H})$.

Example 4. Let a model $\mathcal{M}$ consist of two traces $\mathcal{M} = \{h_1 \bullet h_2 \bullet l_1,\ \ h_1 \bullet h_3 \bullet l_2\}$. Is $\mathcal{M}$ secure wrt. policy $\mathcal{P}ol = (\mathtt{L}, \mathtt{H})$? Intuitively, the answer should be negative as an attacker can associate h_2 with observation l_1 and h_3 with observation l_2. Applying Def. 1, with $0 \leq i \leq 2$, we obtain

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow \mathtt{H}} = \mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow \mathtt{H}} = \{h_1, h_2, h_3\}$
- $\mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathtt{H}} = \{h_1 \bullet h_2 \bullet l_1\}_{\downarrow \mathtt{H}} = \{h_1, h_2\}$
- $\mathcal{K}(\tau_2, 3, \mathtt{L})_{\downarrow \mathtt{H}} = \{h_1 \bullet h_3 \bullet l_2\}_{\downarrow \mathtt{H}} = \{h_1, h_3\}$

The program is insecure as $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathtt{H}} \not\preceq \mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathtt{H}}$, i.e., $\{h_1, h_2, h_3\} \not\preceq \{h_1, h_2\}$. Using the same policy, another model $\mathcal{M}' = \{h_1 \bullet l_1, h_2 \bullet l_1\}$ is secure.

Example 5. Consider now $\mathcal{M} = \{h_1 \bullet h_1 \bullet l_1, h_1 \bullet l_2\}$ wrt. security policy $\mathcal{P}ol = (\mathtt{L}, \mathtt{H})$. Applying Def. 1, the model is secure. However, there may be cases where

$\mathcal{M}$ is considered insecure. For instance, a low user $\mathtt{L}$ may only be interested in knowing when exactly the system is logging his actions, so that an attack can be performed stealthy. To capture these cases it is enough to consider a policy $\mathcal{P}ol_1 = (\mathtt{L}, \mathit{Mul}(\mathtt{H}))$ or $\mathcal{P}ol_2 = (\mathtt{L}, \mathit{Seq}(\mathtt{H}))$. Let $i \in \{0, 1\}$, then

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet h_1, h_1\}$
- $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet h_1, h_1\}$
- $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet l_2\}_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1\}$
- $\mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet h_1\}$

Clearly, $\mathcal{K}(\tau_2, 1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} \not\preceq \mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})}$ as $\{h_1 \bullet h_1, h_1\} \not\preceq \{h_1\}$.

It is worth noticing that protecting $\mathtt{H}$ and $\mathit{Mul}(\mathtt{H})$ is of little interest for reactive systems that may receive inputs on the same channel multiple times. The following program is considered secure wrt. both $(\mathtt{L}, \mathtt{H})$ and $(\mathtt{L}, \mathit{Mul}(\mathtt{H}))$.

$$P ::= \begin{bmatrix} \mathbf{in}(c, x) \\ \mathbf{if}\ x \geq 0\ \mathbf{then\ out}(c', 1) \\ \mathbf{else\ out}(c', 2) \\ \mathbf{in}(c, y) \\ \mathbf{if}\ y \geq 0\ \mathbf{then\ out}(c', 3) \\ \mathbf{else\ out}(c', 4) \end{bmatrix}$$

Indeed, if c is a high channel and c' is a low channel, then $\mathcal{M}_P = \{h_1 \bullet l_1 \bullet h_2 \bullet l_3, h_2 \bullet l_2 \bullet h_1 \bullet l_4\}$ is secure wrt. both policies. However, when an attacker observes l_1 he knows h_1, i.e., the first high input on c was positive, and similarly, when an attacker observes l_3 he knows h_2, i.e., the second high input was positive as well. Hence, the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ is needed to rule out this program.

Example 6. Let $\mathcal{M} = \{h_1 \bullet h_2 \bullet l_1, h_1 \bullet l_2 \bullet h_2\}$ be a program model. $\mathcal{M}$ is secure wrt. policies in previous examples since the sequence of high actions is the same for both traces. However, an attacker observing l_1 knows that $h_1 \bullet h_2$ has occurred, while this is not ensured if he observes l_2. Similar security issues may arise in scenarios discussed in [11]. To capture such flows, we consider a stronger policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ which protects the interleavings between high actions and occurrences of low actions. Let $i \in \{0, 1\}$, then $\mathcal{M}$ is insecure

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)}$
  $= \{h_1 \bullet h_2 \bullet \bot, h_1 \bullet \bot \bullet h_2\}$
- $\mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \{h_1 \bullet h_2 \bullet l_1\}_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \{h_1 \bullet h_2 \bullet \bot\}$
- $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \mathcal{K}(\tau_2, 3, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \{h_1 \bullet \bot \bullet h_2\}$

$\mathcal{K}(\tau_2, 1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} \not\preceq \mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)}$ i.e. $\{h_1 \bullet h_2 \bullet \bot, h_1 \bullet \bot \bullet h_2\} \not\preceq \{h_1 \bullet \bot \bullet h_2\}$

Dynamic Policies and Declassification The security condition in Def. 1 is too strong to be useful in scenarios where high actions are released intentionally. This is typically the case of dynamic policies where information can be downgraded or upgraded with time.

Example 7. Let $\mathcal{M} = \{h_1 \bullet l_1 \bullet h_2 \bullet l_2,\ h_1 \bullet l_3 \bullet h_3 \bullet l_4\}$ be a model with two traces. $\mathcal{M}$ is insecure as the attacker can distinguish h_2 from h_3 after observing, respectively, l_2 and l_4. This is captured by the policy $(\mathtt{L}, \mathtt{H})$.

- $\mathcal{K}(\tau_i, 0, \mathtt{L})_{\downarrow \mathtt{H}} = \mathcal{K}(\tau_i, 1, \mathtt{L})_{\downarrow \mathtt{H}} = \{h_1, h_2, h_3\}$
- $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathtt{H}} = \mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathtt{H}} = \mathcal{K}(\tau_1, 4, \mathtt{L})_{\downarrow \mathtt{H}} = \{h_1, h_2\}$
- $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathtt{H}} = \mathcal{K}(\tau_2, 3, \mathtt{L})_{\downarrow \mathtt{H}} = \mathcal{K}(\tau_2, 4, \mathtt{L})_{\downarrow \mathtt{H}} = \{h_1, h_3\}$

It is clear that $\mathcal{K}(\tau_1, 1, \mathtt{L})_{\downarrow \mathtt{H}} \not\preceq \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathtt{H}}$ as $\{h_1, h_2, h_3\} \not\preceq \{h_1, h_2\}$. However if we declassify $\{h_2\}$ and $\{h_3\}$, resp., at points (t_1, i) and (t_2, i), for $1 \leq i \leq 4$ then the system is secure.

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow \mathtt{H}} \cap \{h_2\} \preceq \mathcal{K}(\tau_1, i+1, \mathtt{L})_{\downarrow \mathtt{H}}$
- $\mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow \mathtt{H}} \cap \{h_3\} \preceq \mathcal{K}(\tau_2, i+1, \mathtt{L})_{\downarrow \mathtt{H}}$

4 Equivalences

In this section we show equivalences between knowledge-based conditions and trace-based conditions from Sect. 2. The first proposition shows that Sep for asynchronous systems is equivalent to knowledge-based condition with security policy $\mathcal{P}ol = (\mathtt{L}, \mathit{Seq}(\mathtt{H}))$.

Proposition 1. *Let $\mathcal{M}$ be the model of an asynchronous program and closed under interleavings of $\tau_{\mathtt{L}}$ and $\tau'_{\mathtt{H}}$. Then $\mathcal{M}$ satisfies Sep iff $\mathcal{M}$ is secure wrt.* $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$.

Proof. We show that $\mathcal{M}$ satisfies Sep iff for all traces $\tau \in \mathcal{M}$, $0 \leq i < |\tau|$, $\mathcal{K}(\tau, i, \mathtt{L})_{\mathit{Seq}(\mathtt{H})} \subseteq \mathcal{K}(\tau, i+1, \mathtt{L})_{\mathit{Seq}(\mathtt{H})}$. ($\Rightarrow$) Suppose $\mathcal{M}$ satisfies Sep. By definition $\forall \tau_1, \tau_2 \in \mathcal{M}$ and $\forall \tau^* \in \ltimes(\tau_{1\mathtt{L}}, \tau_{2\mathtt{H}})$, $\tau^* \in \mathcal{M}$. We show that for all $\tau \in \mathcal{M}$, for all $0 \leq i < |\tau|$, $\mathcal{K}(\tau, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} \subseteq \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})}$. Consider a sequence s^* such that $s^* \in \mathcal{K}(\tau, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})}$ and show that $s^* \in \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})}$. Let $\tau^* \in \mathcal{M}$ be such that $\tau^*_{\downarrow \mathit{Seq}(\mathtt{H})} = s^*$ and $(\tau^*, j) =_{\downarrow \mathtt{L}} (\tau, i)$ and $0 \leq j < |\tau^*|$. We look for a trace $\tau' \in \mathcal{M}$ with $\tau'_{\ltimes(\mathtt{H})} = s^*$ and $(\tau, i+1) =_{\downarrow \mathtt{L}} (\tau, j')$, for some j'. If event $\tau(i{+}1) \in \mathtt{H}$, we pick τ^* and conclude the proof. Otherwise, $\tau(i{+}1) \in \mathtt{L}$. Since Sep holds, it is possible to interleave the low sequence of events up to point $(\tau, i+1)$ with s^* and obtain a trace $\tau'' \in \mathcal{M}$ such that $s^* \in \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})}$ and $(\tau, i+1) =_{\downarrow \mathtt{L}} (\tau'', j'')$, for some j''. ($\Leftarrow$) Assuming closure under interleavings, the claim follows immediately.

At this point the reader may wonder if a stronger policy such as $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ can avoid the assumption of closure under interleavings. The example shows that this is not the case. The main reason is that while separability allows observers to make deductions about future or past occurrences of actions, this is not possible for the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$, which protects all interleavings. On the other hand, if $\mathcal{M}$ lacks some interleavings between sequences of high and low actions and this doesn't affect the initial knowledge of the observer, then the system is considered secure, whilst Sep can still break.

Example 8. The model $\mathcal{M} = \{h_1 \bullet h_2 \bullet l_1 \bullet l_2,\ h_1 \bullet h_2 \bullet l_1 \bullet l_3\}$ does not satisfy Sep, but it is secure wrt. the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ as the observer knows, from knowledge of the initial model, all interleavings, i.e, $\{h_1 \bullet h_2 \bullet \bot \bullet \bot\}$. On the other hand, if $\mathcal{M} = \{h_1 \bullet l_1, l_1 \bullet h_1, h_1 \bullet l_2 \bullet l_3, l_2 \bullet h_1 \bullet l_3, l_2 \bullet l_3 \bullet h_1\}$ and $\mathcal{P}ol = (\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$, then Sep holds. However the epistemic condition does not hold since the sequence $h_1 \bullet \bot \bullet \bot$ is not possible after observing l_1.

The next proposition shows that security wrt. policy $\mathcal{P}ol_1 = (\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ is equivalent to security wrt. $\mathcal{P}ol_2 = (\mathtt{H}, \mathit{Seq}(\mathtt{L}))$.

Proposition 2. *A model $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ iff $\mathcal{M}$ is secure wrt. $(\mathtt{H}, \mathit{Seq}(\mathtt{L}))$.*

The next proposition shows the equivalence between Sep and its epistemic sibling in a synchronous setting.

Proposition 3. *Let $\mathcal{M}$ be the model of a synchronous program. Then $\mathcal{M}$ satisfies SSep iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$.*

It is worth noting that, differently from [8], no additional property on model $\mathcal{M}$ is needed to show the equivalence in Prop. 3. The main reason is that our work considers traces of both finite and infinite length, hence no property as *limit closure* is required.

Proposition 4. *Let $\mathcal{M}$ be closed under interleavings of $\tau_{\mathtt{L}}$ and $\tau'_{\mathtt{HI}}$. Then $\mathcal{M}$ satisfies GNI iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{HI}))$.*

Proposition 5. *A model $\mathcal{M}$ satisfies ND iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{HI}))$.*

Proposition 6. *If a model $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ then $\mathcal{M}$ satisfies NDS.*

Example 9. Consider program P from [18] where a high user transmits a bit z to a low user by sending $z \otimes x$ as input on high channel. Let $c_1 \in \mathcal{O}$ and $c_2, c_3 \in \mathcal{P}$. Then what the low user receives is the exact value of secret z.

$$P ::= x := 0 || 1;\ \mathbf{out}(c_3, x);\ \mathbf{in}(c_2, y);\ \mathbf{out}(c_1, x \otimes y)$$

It can be checked that M_P is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{HI}))$, i.e. ND, and insecure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$. Hence, P does not satisfy NDS.

The following propositions show that the epistemic conditions can be seen as closures over a poset where the ordering relation is given by the security policy. This gives a systematic characterization of security conditions wrt. what is protected and how powerful an attacker is.

Proposition 7. *The security policies $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ are closures over a poset $(\wp(E^*), \preceq)$. In particular, $(\mathtt{L}, \mathtt{H}) \sqsubseteq (\mathtt{L}, \mathit{Mul}(\mathtt{H})) \sqsubseteq (\mathtt{L}, \mathit{Seq}(\mathtt{H})) \sqsubseteq (\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$.*

Proposition 8. *Let $\mathcal{M}$ be a model and $\mathcal{P}ol_1 = (\mathcal{O}_1, \mathcal{P}_1)$, $\mathcal{P}ol_2 = (\mathcal{O}_2, \mathcal{P}_2)$ be security policies. If $\mathcal{P}ol_1 \sqsubseteq \mathcal{P}ol_2$, i.e., $\mathcal{O}_1 \sqsubseteq \mathcal{O}_2$ and $\mathcal{P}_1 \sqsubseteq \mathcal{P}_2$, then $\mathcal{M}$ is secure wrt. $\mathcal{P}ol_2$ if $\mathcal{M}$ is secure wrt. $\mathcal{P}ol_1$.*

We conclude this section by showing equivalence between nondeducibility on outputs and its epistemic peer condition. The main advantage of NDO is that it allows information flow from low user input channels to high output channels and, at the same time it protects sequences of high actions interleaved with low inputs. To express such requirements, we make use of the release policy which allows a low user to declassify information about sequences of low user inputs and high actions, i.e., $Seq(\mathtt{LI}+\mathtt{H})$, given the low input sequence $\mathtt{LI}$ of the current trace, namely, $\mathcal{R}(\tau, i, \mathtt{LI}) = \{\tau^* \in \mathcal{M} | \tau =_{\mathtt{LI}} \tau^*\}$. The other low inputs, *system inputs* in [11], are modeled as internal nondeterminism.

Proposition 9. *Consider a program model $\mathcal{M}$. Then $\mathcal{M}$ satisfies NDO iff* $\forall \tau, i,\ \mathcal{K}(\tau, i, \mathtt{L})_{\downarrow Seq(\mathtt{LI}+\mathtt{H})} \cap \mathcal{R}(\tau, i, \mathtt{LI})_{\downarrow Seq(\mathtt{LI}+\mathtt{H})} \preceq \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow Seq(\mathtt{LI}+\mathtt{H})}$.

The following example from [19], shows that our condition handles correctly information from $\mathtt{LI}$ to $\mathtt{HO}$. The key point here is the use of release policy to break the symmetry inherent in nondeducibility-like conditions.

Example 10. Consider P with low channels c_1, c_3 and high channels c_2, c_4.

$$P ::= \mathbf{in}(c_1, x); \mathbf{out}(c_2, x); \mathbf{out}(c_3, x); \mathbf{in}(c_4, y)$$

If $\mathbf{in}(c_1, x)$ is a low user input, the program can be considered secure as nothing about high actions is revealed. However, if $\mathbf{in}(c_1, x)$ is a *system* input, then the value is incorrectly transmitted to the low user through $out(c_3, x)$. The security condition captures both cases.

5 A Logic for Information Flow

In this section we express the security conditions in Sect. 2 in terms of a logic of knowledge and time. We consider the framework of multi-agent systems [20] and extend the logic presented in [6] to reason about possibilistic security conditions.

5.1 Knowledge in Multi-agent Systems

The framework of multi-agent systems allows reasoning about knowledge and time in a distributed system where different agents (users, processes) interact with each other. The system consists of a set of agents $Ag = \{a_i\}_{i=1}^k$ which have local state L_i at a given point in time. A special agent E, with local state L_E, models the environment where the distributed system runs. The global state consists of a tuple of local states, i.e., $G = (L_E, L_1, \cdots, L_k)$. A *run* is a sequence of global states over discrete time. An *interpreted system* [20] is a pair $\mathcal{I} = (R, \Pi)$ of runs R and interpretation function Π over a set Φ of atomic propositions. Program models can be associated with interpreted systems. Given a point (τ, i), then $L_E = (trace(\tau, i))$. The local state of an agent a who observes $\mathcal{O}$ is $L_a = (trace(\tau, i)_{\downarrow \mathcal{O}})$. Then, the global state of a system with k agents who observe $\mathcal{O}_1, \cdots, \mathcal{O}_k$ is $G = ((trace(\tau, i)), (trace(\tau, i)_{\downarrow \mathcal{O}_1}), \cdots, (trace(\tau, i)_{\downarrow \mathcal{O}_k}))$.

The atomic propositions in Φ describe basic facts about the model. In our context, the facts refer to actions on channels. The interpretation function $\Pi(p)(G)$ assigns a truth value to all $p \in \Phi$. To define knowledge, an interpreted system $\mathcal{I} = (R, \Pi)$ is associated with a *Kripke structure* $\mathcal{M}_{\mathcal{I}} = (G, \Pi, \{K_i\}_{i=1}^{k})$ where G and Π are as before and K_i is a binary relation over G. In particular, $K_i(G) = \{G' | G \sim_i G'\}$, where $G \sim_i G'$ if L_i is the same in both states.

5.2 Temporal Epistemic Logic with Past

We now present a logic with temporal and epistemic operators to reason about security properties in a syntactical manner. Let $\Phi = \{\mathbf{in}(c, v), \mathbf{out}(c', v') | c, c' \in \mathit{Chan}$ and $v, v' \in \mathit{Val}\}$ be the set of atomic propositions. We consider a language containing Φ and closed off under conjunction, negation, knowledge operators K_a, temporal operators *Next* X and *Until* U and past time operators *Initially* I, *Previous* Y and *Since* S. Let $p \in \Phi$,

Definition 10 (Syntax of $\mathcal{L}_{KPLTL}$).
The language $\mathcal{L}_{KPLTL}$ of formulas ϕ, ψ in linear time temporal epistemic logic with past is given as follows:

$$\phi, \psi ::= p \mid \phi \wedge \psi \mid \neg\phi \mid K_a\phi \mid X\phi \mid \phi U \psi \mid I\phi \mid Y\phi \mid \phi S \psi$$

The operator K_a is the epistemic knowledge operator. $K_a\phi$ holds if ϕ holds in any point equivalent to the current point of agent a. The formula $\phi U \psi$ holds if ψ holds in a future point and ϕ holds until reaching that point. Dually, the formula $\phi S \psi$ holds if ψ was true once in the past and ϕ has been true ever since. $I\phi$ holds if ϕ is true initially, while $X\phi$ ($Y\phi$) hold if ϕ is true at the next (previous) point. Various connectives are definable in $\mathcal{L}_{KPLTL}$ including boolean operators such as $\vee$ and $\rightarrow$, the truth constants *tt* and *ff*, the epistemic possibility operator $L_a\phi$ meaning that ϕ holds for at least one epistemically equivalent point, the future (past) operator $F\phi$ ($O\phi$) requiring ϕ to eventually hold in the future (past), the always (historically) operator $G\phi$ ($H\phi$) meaning that ϕ holds in any future (past) state, and the weak until $\phi W \psi$ which does not require ψ to eventually hold. Finally, the operator K_G is the group knowledge operator, the formula $K_G\phi$ holds if the combined knowledge of G members implies ϕ. The logic is sufficiently expressive to specify all information flow policies in Sect. 1.

Definition 11 (Satisfaction). *Fig. 1 defines the satisfaction relation $\mathcal{M}, (\tau, i) \models \phi$ between points in a model $\mathcal{M}$ and $\mathcal{L}_{KPLTL}$ formulas. In particular, satisfaction relative to model $\mathcal{M}$ is defined as $\mathcal{M} \models \phi$ iff $\forall \tau \in \mathcal{M},\ \mathcal{M}, (\tau, 0) \models \phi$.*

At this point we have all ingredients to characterize the security condition in Sect. 2 by means of $\mathcal{L}_{KPLTL}$ formulas. First notice that the set $\mathcal{K}(\tau, i, \mathcal{O})$ represents all traces that an observer $\mathcal{O}$ considers possible at point (τ, i), which corresponds to the uncertainty of the observer at that point. Given a policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ and a formula ϕ specifying properties of $\mathcal{P}$, we show that ϕ is possible at any point (τ, i) by means of the operator $L_a\phi$. To avoid complications

$\mathcal{M}, (\tau, i) \models p$ iff $\tau(i) = p$

$\mathcal{M}, (\tau, i) \models \phi \wedge \psi$ iff $(\tau, i) \models \phi$ and $(\tau, i) \models \psi$

$\mathcal{M}, (\tau, i) \models \neg\phi$ iff $(\tau, i) \not\models \phi$

$\mathcal{M}, (\tau, i) \models X\phi$ iff $i + 1 \leq len(\pi)$ and $(\tau, i + 1) \models \phi$

$\mathcal{M}, (\tau, i) \models \phi U \psi$ iff $\exists j : i \leq j \leq len(\tau)$ such that $(\tau, j) \models \psi$ and $\forall k : i \leq k < j,\ (\tau, k) \models \phi$

$\mathcal{M}, (\tau, i) \models I\phi$ iff $(\tau, 0) \models \phi$

$\mathcal{M}, (\tau, i) \models Y\phi$ iff $i - 1 \geq 0$ and $(\tau, i - 1) \models \phi$

$\mathcal{M}, (\tau, i) \models \phi S \psi$ iff $\exists j : 0 \leq j \leq i$ such that $(\tau, j) \models \psi$ and $\forall k : j < k \leq i,\ (\tau, k) \models \phi$

$\mathcal{M}, (\tau, i) \models K_a\phi$ iff $\forall \tau' \in \mathcal{M}, \forall (\tau', i') \in \tau'$ s.t. $trace_a(\tau, i) = trace_a(\tau', i'),\ (\tau', i') \models \phi$

Fig. 1. Satisfaction at trace points

due to observations at the limit, we assume that the models are limit closed. Namely, all properties of $\mathcal{P}$ can be captured by finitely many observations in $\mathcal{O}$. The view of agent a (group G) observing $\mathcal{O}_a$ ($\mathcal{O}_G = \bigcup_{a \in G} \mathcal{O}_a$) is defined as $trace_a(\tau, i) = trace(\tau, i)_{\downarrow \mathcal{O}}$ ($trace_G(\tau, i) = trace(\tau, i)_{\downarrow \mathcal{O}_G}$). Then the following theorem relates the semantic conditions to the syntactical ones.

Theorem 1. *Consider a model $\mathcal{M}$ and a security policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$. Let also $\phi_1, \cdots, \phi_n$ be a set of $\mathcal{L}_{KPLTL}$ formulas encoding information to be protected, i.e. $\mathcal{M}_{\downarrow \mathcal{P}}$. Then $\mathcal{M}$ is secure wrt. $\mathcal{P}ol$ iff $\mathcal{M} \models \bigwedge_{i=1}^{n} GL_a\phi_i$.*

Finally it remains to show that the logic can be used describe different protection policies, as defined in Sect. 3. In particular, each element in $\mathcal{M}_{\downarrow \mathcal{P}}$ can be encoded using the logic in Fig. 1. Let $p \in \Phi$, then we define auxiliary formulas: $Occ(p) = (O\,p \vee F\,p)$ meaning that p eventually holds at a (past or future) point, $SV(c) = \bigvee_{v \in Val} e(c, v)$ meaning that an action has happened on channel c, $Occ(p, i) = O\,(I\ tt\ \wedge F\,(p \wedge X\,F\,(p \wedge X\,F\,(p \wedge \cdots))))$ meaning that p is true in at least i different points in the current trace and a happens-before formula $HB(p, q) = O\,(\neg q U (p \wedge F\,q)) \vee F\,(\neg q U (p \wedge F\,q))$ meaning that p holds before q at some point in the current trace. Moreover, auxiliary formulas can be combined to express facts ϕ that occur infinitely many times by using the formula $Inf(\phi) = G\,F\phi$.

Proposition 10. *Consider a model $\mathcal{M}$ and a policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$. Then the following variants of $\mathcal{P}$ can be encoded in $\mathcal{L}_{KPLTL}$,*

- *Set*(H)*: $\mathcal{M} \models \bigwedge_{i=1}^{n} GL_a\,Occ(e_i(c_i, v_i))$, where $c_i \in \mathcal{P}$*
- *Mul*(H)*: $\mathcal{M} \models \bigwedge_{i=1}^{n} GL_a\,Occ(e_i(c_i, v_i), k_i)$, where $c_i \in \mathcal{P}$ and k_i is the multiplicity of e_i*
- *Seq*(H)*: $\mathcal{M} \models L_a \bigwedge_{j=2}^{k} HB(e_{j-1}, e_j)$ for all high sequences ϕ_i*
- *Seq*(H $\bot$) *iff $\mathcal{M} \models GL_a \bigwedge_{j=2}^{k} HB(p_{j-1}, p_j)$, for all sequences ϕ_i such that $p_j = SV(c)$ if $e_j(c, v) = p_j$ and $c \in \mathcal{O}$, or $p_j = e_j(c, v)$ if $c \in \mathcal{P}$*

6 Related Work and Conclusions

We are not the first to examine connection between trace-based and knowledge-based information flow properties. A closely related work is that of Halpern and O'Neill [8], who also consider formal definitions of secrecy in multiagent systems. They show how SEP and GNI fit in the epistemic framework, both in synchronous and asynchronous setting. In particular, they define knowledge as a set of points that an agent considers possible based on his local state. By contrast, this paper defines knowledge as the set of global traces that an agent considers possible based on his local observations. This allows us to give security conditions which are closer to what is used in language-based security [21, 10, 7]. Furthermore, the logic we present here captures directly the security properties of traces.

Recently, knowledge-based conditions for information flow have been popular in language-based security. Several works [4, 22] explore epistemic conditions for relational and interactive models, although in a synchronous setting only. Different issues related to declassification [6] and attack models [10] have been considered using epistemic security conditions. In [23] Sabelfeld and Mantel discuss information flow security for distributed programs and point out finer-grained sources of leaks due to encryption, environment totality and timing. All these subtle flows can be accurately captured by the security condition presented here.

The majority of verification techniques for information flow properties rely on security type systems, as in [5]. However, several model checking approaches, which were recently proposed [24, 25], define fragments of logics for which verification is feasible. An interesting future direction would be to devise fragments of $\mathcal{L}_{KPLTL}$ for which model checking has low complexity.

In conclusion, we have discussed several possibilistic information flow conditions and showed how knowledge-based account can be used to specify these conditions, both semantically and syntactically. The advantage of using epistemic logic is that it can accurately express complex policies and provide a clear separation between the code and the security annotations. However, complexity of verification can be very high for large programs. Different remedies to this issue require further investigation. First, abstraction techniques at the program level can be used to obtain smaller models which can be easy to verify. Second, distributed system properties, such as asynchrony, order preservation or lossiness can be used to decompose the epistemic formulas into simpler ones, which are easier to verify. Finally, a hybrid verification which combines type checking and model checking is another path that deserves further exploration.

Acknowledgements. Thanks to Mads Dam for many valuable discussions. This work was supported by SSF-funded project PROSPER ($\mathcal{N}^{\underline{o}}$ RIT10-0069).

References

[1] Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proc. IEEE Symp. on Security and Privacy, Los Alamitos, CA, IEEE Comp. Soc. Press (1982)

11–20
[2] Denning, D.E., Denning, P.: Certification of programs for secure information flow. Communications of the ACM **20**(7) (1977) 504–513
[3] Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE J. on selected ares in communications **21**(1) (2003) 5–19
[4] O'Neill, K.R., Clarkson, M.R., Chong, S.: Information-flow security for interactive programs. In: CSFW. (2006) 190–201
[5] Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. Journal of Computer Security **4**(2,3) (1996) 167–187
[6] Balliu, M., Dam, M., Le Guernic, G.: Epistemic Temporal Logic for Information Flow Security. In: PLAS. (2011)
[7] Balliu, M., Dam, M., Guernic, G.L.: Encover: Symbolic exploration for information flow security. In: CSF. (2012) 30–44
[8] Halpern, J.Y., O'Neill, K.R.: Secrecy in multiagent systems. ACM Trans. Inf. Syst. Secur. **12**(1) (2008)
[9] Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: IEEE Symposium on Security and Privacy. (2007) 207–221
[10] Askarov, A., Chong, S.: Learning is change in knowledge: Knowledge-based security for dynamic policies. In: CSF. (2012) 308–322
[11] Guttman, J.D., Nadel, M.E.: What needs securing. In: CSFW. (1988) 34–57
[12] Jhala, R., Majumdar, R.: Software model checking. ACM Comput. Surv. **41**(4) (2009)
[13] Balliu, M., Le Guernic, G.: Encover (June 2012) Software release.
[14] Balliu, M.: A logic for information flow analysis of distributed programs (extended abstract). Technical report, KTH Royal Institute of Technology
[15] Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. J. of Computer Security (2007)
[16] Mclean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: In Proc. IEEE Symposium on Security and Privacy. (1994) 79–93
[17] Sutherland, D.: A model of information. In: 9th National Computer Security Conference. (1986)
[18] Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: IEEE Symposium on Security and Privacy. (1990) 144–161
[19] McLean, J.: Security models and information flow. In: In Proc. IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1990) 180–187
[20] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about knowledge. MIT Press, Cambridge, Mass. (1995)
[21] Askarov, A., Myers, A.C.: Attacker control and impact for confidentiality and integrity. Logical Methods in Computer Science **7**(3) (2011)
[22] Askarov, A., Sabelfeld, A.: Tight enforcement of information-release policies for dynamic languages. In: CSF. (2009) 43–59
[23] Sabelfeld, A., Mantel, H.: Securing communication in a concurrent language. In: SAS. (2002) 376–394
[24] Alur, R., Cerný, P., Chaudhuri, S.: Model checking on trees with path equivalences. In: TACAS. (2007) 664–678
[25] Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: VMCAI. (2012) 169–185