A Logical Framework for History-Based Access Control and Reputation Systems

Philip W. L. Fong

Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
pwlfong@ucalgary.ca

CPSC 601.40 (Fall 2016)
Motivating Example

- eBay bidding
  - Winning bidder sends payment
  - Seller receives payment, confirms reception, and ships item
  - Optionally, both buyer and seller may leave feedback (e.g., positive, neutral, negative)
Parties engage in transactions
Each transaction is structured according to some protocol
Want to use outcomes of previous transactions to assess the trustworthiness of the parties.
Novelty: structuring this in terms of History-Based Access Control
Outline

1. Observations as Events
2. Policy Language
3. Enforcement
Outline

1. Observations as Events
2. Policy Language
3. Enforcement
**Definition 2.1** (Event structure). An *event structure* is a triple $ES = (E, \leq, \#)$ consisting of a set $E$, and two binary relations on $E$: $\leq$ and $\#$. The elements $e \in E$ are called *events*, and the relation $\#$, called the *conflict relation*, is symmetric and irreflexive. The relation $\leq$ is called the *(causal) dependency relation*, and partially orders $E$. The dependency relation satisfies the following axiom, for any $e \in E$:

$$\text{the set } [e] \overset{(\text{def})}{=} \{ e' \in E \mid e' \leq e \} \text{ is finite.}$$

The conflict- and dependency-relations satisfy the following “transitivity” axiom for any $e, e', e'' \in E$,

$$(e \# e' \text{ and } e' \leq e'') \text{ implies } e \# e''.$$  

Two events are *independent* if they are not in either of the two relations.
Example of an Event Structure

confirm \sim\sim time-out

\begin{itemize}
  \item pay \sim\sim ignore
  \item positive \sim\sim neutral \sim negative
\end{itemize}

- An event structure modelling the buyer’s observations in eBay. (Immediate)
  Conflict is represented by \sim and dependency by \rightarrow.
- *Intuition:* how a single protocol run looks like
  - Conflict: can’t have both
  - Dependency: one must occur before the other
Definition 2.2 (Configuration). Let $ES = (E, \leq, \#)$ be an event structure. We say that a subset of events $x \subseteq E$ is a configuration if it is conflict free (C.F.), and causally closed (C.C.). That is, it satisfies the following two properties, for any $d, d' \in x$ and $e \in E$,

\[(\text{C.F.}) \ d \neq d'; \quad \text{and} \quad (\text{C.C.}) \ e \leq d \Rightarrow e \in x.\]

- **Intuition:** the “state” of a protocol run
Notations for Configurations

**Notation 2.1.** $C_{ES}$ denotes the set of configurations of $ES$, and $C_{ES}^0 \subseteq C_{ES}$ the set of *finite* configurations. A configuration is said to be *maximal* if it is maximal in the partial order $(C_{ES}, \subseteq)$. Also, if $e \in E$ and $x \in C_{ES}$, we write $e \# x$, meaning that $\exists e' \in x. e \# e'$. Finally, for $x, x' \in C_{ES}$, $e \in E$, define a relation $\rightarrow$ by $x \xrightarrow{e} x'$ iff $e \notin x$ and $x' = x \cup \{e\}$. If $y \subseteq E$ and $x \in C_{ES}$, $e \in E$ we write $x \xrightarrow{e} y$ to mean that either $y \notin C_{ES}$ or it is not the case that $x \xrightarrow{e} y$.

- **Intuition:**
  - $C_{ES}$ is the set of all “states”
  - $C_{ES}^0$ is the set of all states reachable in finitely many transitions
  - $e \# x$: Event $e$ is not applicable in state $x$
  - Transition relation: $x \xrightarrow{e} x'$
  - Maximal: the protocol has completed
Example (Examples of Configuration)

{pay, positive}
Example (Examples of Configuration)

- $\{\text{pay, positive}\}$
- $\emptyset$: initial state
### Example (Examples of Configuration)

- \{\text{pay}, \text{positive}\}
- \emptyset: initial state
- \{\text{pay}, \text{confirm}, \text{positive}\}: maximal
Examples

Example (Examples of Configuration)
- \{\text{pay, positive}\}
- \emptyset: initial state
- \{\text{pay, confirm, positive}\}: maximal

Example (Negative Examples for Configurations)
- \{\text{confirm}\} (Why?)
Examples

Example (Examples of Configuration)

- \{pay, positive\}
- \emptyset\): initial state
- \{pay, confirm, positive\}: maximal

Example (Negative Examples for Configurations)

- \{confirm\} ↩ not causally closed
Examples

Example (Examples of Configuration)
- \{\text{pay, positive}\}
- \emptyset: initial state
- \{\text{pay, confirm, positive}\}: maximal

Example (Negative Examples for Configurations)
- \{\text{confirm}\} ← not causually closed
- \{\text{pay, confirm, positive, negative}\} (Why?)
Examples

Example (Examples of Configuration)
- \{\text{pay, positive}\}
- \emptyset: initial state
- \{\text{pay, confirm, positive}\}: maximal

Example (Negative Examples for Configurations)
- \{\text{confirm}\} \leftarrow \text{not causually closed}
- \{\text{pay, confirm, positive, negative}\} \leftarrow \text{not conflict free}
Local Interaction History

**DEFINITION 2.3 (Local interaction history).** Let $ES$ be an event structure, and define a local interaction history in $ES$ to be a sequence of finite configurations, $h = x_1 x_2 \cdots x_n \in \mathcal{C}_{ES}^0$*. The individual components $x_i$ in the history $h$ will be called *sessions*.

- **Intuition:**
  - A sequence of protocol runs that have been initiated in the past.
  - Each member of the sequence is a configuration (i.e., states of protocol sessions)
  - The protocol sessions need not have been completed
DEFINITION 2.4 (Interface). Define an operation \( \text{new} : C_{ES}^0 \rightarrow C_{ES}^0 \) by \( \text{new}(h) = h\emptyset \). Define also a partial operation \( \text{update} : C_{ES}^0 \times E \times \mathbb{N} \rightarrow C_{ES}^0 \) as follows. For any \( h = x_1 x_2 \cdots x_i \cdots x_n \in C_{ES}^0 \), \( e \in E \), \( i \in \mathbb{N} \), \( \text{update}(h, e, i) \) is undefined if \( i \notin \{1, 2, \ldots, n\} \) or \( x_i \not\rightarrow x_i \cup \{e\} \). Otherwise

\[
\text{update}(h, e, i) = x_1 x_2 \cdots (x_i \cup \{e\}) \cdots x_n.
\]

- **Intuition:**
  - These are operations that can be performed over a history
  - \( \text{new} \): initiates a new protocol session
  - \( \text{update} \): one more event occurring in an existing protocol session in the history
Example

- \texttt{update}\{\texttt{pay}\}, \texttt{neutral}, 1\}
  
  \[ = ? \]
Example

\( \text{update}\{\text{pay}\}, \text{neutral}, 1 \) = \{\text{pay}, \text{neutral}\}
Example

- \textbf{update}\{\{pay\}, neutral, 1\} = \{pay, neutral\}
- \textbf{update}\{new\{\{pay\}\}, ignore, 2\} = ?
**Example**

- \textbf{update}\{{pay},\text{neutral},1\} = \{\text{pay,neutral}\}

- \textbf{update(new}\{{pay}\},\text{ignore},2\} = \{\text{pay}\}\{\text{ignore}\}
Given an event structure $ES = (E, \leq, \#)$, the policy language $\mathcal{L}(ES)$ is given below:

$$\psi ::= e \mid \Diamond e \mid \psi_0 \land \psi_1 \mid \neg \psi \mid X^{-1}\psi \mid \psi_0 \S \psi_1.$$ 

where $e \in E$.

- Both $e$ and $\Diamond e$ are atomic propositions.
- $\Diamond e$ means “$e$ is still possible”. $\Diamond$ is not a temporal operator.
Suppose $h = x_1 x_2 \cdots x_N$.

$(h, i) \models e$  
iff $1 \leq i \leq N$ and $e \in x_i$

$(h, i) \models \Diamond e$  
iff $1 \leq i \leq N \Rightarrow e \not\in x_i$

$(h, i) \models \psi_0 \land \psi_1$  
iff $(h, i) \models \psi_0$ and $(h, i) \models \psi_1$

$(h, i) \models \neg \psi$  
iff $(h, i) \not\models \psi$

$(h, i) \models X^{-1} \psi$  
iff $i > 1$ and $(h, i - 1) \models \psi$

$(h, i) \models \psi_0 S \psi_1$  
iff $\exists j \leq i. [(h, j) \models \psi_1$ and $\forall k. (j < k \leq i \Rightarrow (h, k) \models \psi_0)]$. 
Semantics: \( h \models \psi \)

- We write \( h \models \psi \) iff \( (h, |h|) \models \psi \).
Derived Forms

false = e \land \neg e

true = \neg false

\psi_0 \lor \psi_1 = \neg (\neg \psi_0 \land \neg \psi_1)

\psi_0 \rightarrow \psi_1 = \neg \psi_0 \lor \psi_1

F^{-1}(\psi) = true S \psi

G^{-1}(\psi) = \neg F^{-1}(\neg \psi)
Example: eBay

Example (Bidding Policy)

“Only bid on auctions run by a seller that has never failed to send goods for won auctions in the past.”

\[ \psi_{\text{bid}} \overset{\text{def}}{=} \neg F^{-1}(\text{time-out}) \]
Example: eBay

Example (Bidding Policy)
“Only bid on auctions run by a seller that has never failed to send goods for won auctions in the past.”

\[ \psi^{\text{bid}} \overset{\text{def}}{=} \neg F^{-1}(\text{time-out}) \]

Example (Another Bidding Policy)
“... and the seller has never provided negative feedback in auctions where payment was made.”

\[ \psi^{\text{bid}} \overset{\text{def}}{=} \neg F^{-1}(\text{time-out}) \land G^{-1}(\text{negative} \rightarrow \text{ignore}) \]
Outline

1. Observations as Events
2. Policy Language
3. Enforcement
A reference monitor that:
- Protects a resource by a policy formula $\psi$.
- Tracks local interaction history $h$.
- Grants access request when:

$$h \models \psi$$

where $h = x_1 x_2 \cdots x_N$.
The reference monitor is modeled as an object DMC supporting the following operations:

- **constructor**
  - Argument 1: \( ES = (E, \leq, \#) \)
  - Argument 2: \( \psi \)
  - Initialize \( h \) to empty sequence \( \epsilon \)

- **new()**
  - Create a new protocol session at the end of the local interaction history \( h \).

- **update\((e, i)\)**
  - Add event \( e \) to \( x_i \) (the configuration of protocol session \( i \)).

- **check()**
  - Check \( h \models \psi \)
Naive Implementation

- Naive implementation:
  - Store the entire history $h$
  - Each time `check()` is invoked, go through the entire history to model check $\psi$ (i.e., following the semantic rules).

- Problem:
  - Has to store a potentially long $h$
  - Takes a long time to model check
Observations

1. No need to store the entire $h$
2. No need to go through $h$ to model check $\psi$
Active History

- History $h$ can be “factorized” into two parts:

$$h = \underbrace{x_1 x_2 \cdots x_M}_{\text{longest maximal prefix}} \cdot \underbrace{y_1 y_2 \cdots y_K}_{\text{active history}}$$

- The longest maximal prefix is the longest prefix of $h$ such that every protocol session in it is maximal.
- The remaining sessions may or may not be maximal.
Active History

- History $h$ can be “factorized” into two parts:

$$h = x_1 x_2 \cdots x_M \cdot y_1 y_2 \cdots y_K$$

  - longest maximal prefix
  - active history

- The longest maximal prefix is the longest prefix of $h$ such that every protocol session in it is maximal.
- The remaining sessions may or may not be maximal.
- There is no need to store the longest maximal prefix $x_1 x_2 \cdots x_M$.
- Their influence on the satisfiability of $h$ will not change.
- Store only the actual history $y_1 y_2 \cdots y_K$ as the update and new operations will only modify
Maintains $K + 1$ bit vectors $B_0[\cdot], B_1[\cdot], \ldots, B_K[\cdot]$

Each bit vector $B_k[\cdot]$ is indexed by subformulas of $\psi$.

Subformulas of $\psi$ is ordered as $\psi_0, \psi_1, \psi_2, \ldots, \psi_n$ such that:

1. $\psi_0 = \psi$
2. if $\psi_i$ is a proper subformula of $\psi_j$ then $i > j$.

Invariants:

$B_k[\psi_j] = 1$ iff:

$$(x_1x_2\cdots x_M \cdot y_1y_2\cdots y_k, M + k) \models \psi_j$$

Special case: $B_0[\cdot]$ captures the satisfiability of subformulas for the longest maximal prefix.
Initialization of DMC

- Initialize actual history to empty sequence $\epsilon$.
- $M := 0; \ K = 0$
- Compute $B_0[\cdot]$
Simply look up $B_K[\psi_0]$. 
2 Problematic Operations

- **update** \((e, i)\)
  - Let \(j = i - M\)
  - Add \(e\) to \(y_j\)
  - Recompute \(B_j[:], B_{j+1}[:], \ldots, B_K[:]\).

- **new()**
  - After adding a new session to the end of active history, need to recompute \(B_K[:]\).

Both recomputation can be expensive.
Fixed-Point Computation

- \((h, k) \models X^{-1}\psi\) iff \((h, k - 1) \models \psi\)
- \((h, k) \models \phi S \psi\) iff either
  1. \((h, k) \models \psi\), or
  2. \((h, k) \models \phi\) and \((h, k - 1) \models \phi S \psi\)

We can compute the value of \(B_k[\psi_j]\) by either:

1. looking up entries from \(B_{k-1}\), or
2. looking up \(B_k[\psi_i]\) for some \(\psi_i\) where \(i > j\) (i.e., \(\psi_i\) is a proper subformula of \(\psi_j\)).

Implication: Dynamic Programming!
### Fixed-Point Computation

1. \((h, k) \models X^{-1}\psi\) iff \((h, k - 1) \models \psi\)
2. \((h, k) \models \phi S \psi\) iff either
   1. \((h, k) \models \psi\), or
   2. \((h, k) \models \phi\) and \((h, k - 1) \models \phi S \psi\)

- We can compute the value of \(B_k[\psi_j]\) by either:
  1. looking up entries from \(B_{k-1}\), or
  2. looking up \(B_k[\psi_i]\) for some \(\psi_i\) where \(i > j\) (i.e., \(\psi_i\) is a proper subformula of \(\psi_j\)).
Fixed-Point Computation

We can compute the value of $B_k[\psi_j]$ by either:

1. looking up entries from $B_{k-1}$, or
2. looking up $B_k[\psi_i]$ for some $\psi_i$ where $i > j$ (i.e., $\psi_i$ is a proper subformula of $\psi_j$).

**Implication:** Dynamic Programming!
Recomputing $B_k[\cdot]$

for $\varphi := \psi_n, \psi_{n-1}, \ldots, \psi_0$ do: // from small to big formulas

Case $\varphi = e$:
$B_k[\varphi] := (e \in y_k)$;

Case $\varphi = \diamond e$:
$B_k[\varphi] := (e \not\in y_k)$;

Case $\varphi = \neg \phi$:
$B_k[\varphi] := \neg B_k[\phi]$;

Case $\varphi = \phi_0 \land \phi_1$:
$B_k[\varphi] := B_k[\phi_0] \land B_k[\phi_1]$;

Case $\varphi = X^{-1} \phi$:
$B_k[\varphi] = B_{k-1}[\phi]$;

Case $\varphi = \phi_0 S \phi_1$:
$B_k[\varphi] := B_k[\phi_1] \lor (B_k[\phi_0] \land B_{k-1}[\varphi])$;
Observations as Events
Policy Language
Enforcement

Updating

- $\textbf{update}(e, i)$
  - Let $j = i - M$
  - Add $e$ to $y_j$
  - Recompute $B_j[\cdot], B_{j+1}[\cdot], \ldots, B_K[\cdot]$ in this order.
New Session

- **new()**
  - Append empty session to end of active history
  - Increment $K$
  - Allocate and then initialize $B_K[\cdot]$ by the recomputation algorithm.
Theorem

- Initializing DMC takes $O(|\psi|)$.  
- $\text{DMC.check}()$ is $O(1)$.  
- $\text{DMC.new}()$ is $O(|\psi|)$.  
- $\text{DMC.update}(e, i)$ is $O((K - j + 1) \times |\psi|)$ where $j = i - M$.  
- The space requirement is $O(K \times |\psi| + |E| \times |C_{ES}|)$.  