A Framework for Concrete Reputation-Systems

... now with Applications to History-Based Access Control

Karl Krukow  Mogens Nielsen  Vladimiro Sassone

BRICS, Department of Computer Science
University of Aarhus, Denmark

\(\pi\lambda\) Seminar
Access Control

With interconnected and open-ended systems, access control becomes increasingly relevant.

Correctness

If entity $p$ gets access to resource $r$ then $p$ is “authorized” to access $r$.

Different mechanisms provide different notions of “authorized.”

- **Identity-based for centralized systems**: e.g., *Access Control Matrices* - $p$ is authorized to access $r$ if entry $(p, r)$ is true.
- **Identity-based for decentralized systems**: e.g., *Public Key Digital Signatures* - $p$ is authorized to access $r$ if $p$ can sign with key $k_p$.
- **Credential-based for decentralized systems**: e.g., *Traditional Trust Management* - $p$ using public key $pk_p$ is authorized if it carries a certificate from an appropriate authority.
Access Control

- With interconnected and open-ended systems, access control becomes increasingly relevant.

Correctness

If entity $p$ gets access to resource $r$ then $p$ is “authorized” to access $r$.

Different mechanisms provide different notions of “authorized.”

- Identity-based for centralized systems: e.g., Access Control Matrices - $p$ is authorized to access $r$ if entry $(p, r)$ is true.
- Identity-based for decentralized systems: e.g., Public Key Digital Signatures - $p$ is authorized to access $r$ if $p$ can sign with key $k_p$.
- Credential-based for decentralized systems: e.g., Traditional Trust Management - $p$ using public key $pk_p$ is authorized if it carries a certificate from an appropriate authority.
Access Control

- With interconnected and open-ended systems, access control becomes increasingly relevant.

Correctness

If entity $p$ gets access to resource $r$ then $p$ is “authorized” to access $r$.

Different mechanisms provide different notions of “authorized.”

- Identity-based for centralized systems: e.g., Access Control Matrices - $p$ is authorized to access $r$ if entry $(p, r)$ is true.
- Identity-based for decentralized systems: e.g., Public Key Digital Signatures - $p$ is authorized to access $r$ if $p$ can sign with key $k_p$.
- Credential-based for decentralized systems: e.g., Traditional Trust Management - $p$ using public key $pk_p$ is authorized if it carries a certificate from an appropriate authority.
Reputation Systems
and dynamic trust management.

- Idea of reputation
  - Behaviour-based: an entity’s behaviour in past interactions determine its privilege in future ones.
  - Relevant for large decentralized systems (often) with multiple interactions.
  - What does it mean for an entity in a reputation system to be “authorized”?

- Existing systems provide no “correctness” criteria.
  - Often “reputation information” undergoes heavy abstraction (e.g. Eigentrust and Ebay).

Reputation System Security

If entity $p$ gains access to resource $r$ at time $t$, then the past behaviour of $p$ up until time $t$ satisfies requirement $\psi_r$. 
Reputation Systems and dynamic trust management...

- Idea of reputation
  - Behaviour-based: an entity’s behaviour in past interactions determine its privilege in future ones.
  - Relevant for large decentralized systems (often) with multiple interactions.
  - What does it mean for an entity in a reputation system to be “authorized”?

- Existing systems provide no “correctness” criteria.
  - Often “reputation information” undergoes heavy abstraction (e.g. Eigentrust and Ebay).

Reputation System Security

If entity $p$ gains access to resource $r$ at time $t$, then the past behaviour of $p$ up until time $t$ satisfies requirement $\psi_r$. 
Example: Edjlali et al. [1]:
- Suppose you’ve downloaded what claims to be a new cool browser from some webpage.
- “allow a program to connect to a remote site if and only if it has neither tried to open a local file that it has not created, nor tried to modify a file it has created, nor tried to create a sub-process.”

Our definition of reputation system security fits well with the goals of history-based access control.

Reputation System Security

If entity $p$ gains access to resource $r$ at time $t$, then the past behaviour of $p$ up until time $t$ satisfies requirement $\psi_r$. 
Example: Edjlali et al. [1]:
- Suppose you’ve downloaded what claims to be a new cool browser from some webpage.
- “allow a program to connect to a remote site if and only if it has neither tried to open a local file that it has not created, nor tried to modify a file it has created, nor tried to create a sub-process.”

Our definition of reputation system security fits well with the goals of history-based access control.

Reputation System Security

If entity $p$ gains access to resource $r$ at time $t$, then the past behaviour of $p$ up until time $t$ satisfies requirement $\psi_r$. 
Outline

1. Modelling behavioural information
   - Event Structures as a general model

2. A Simple Policy Language
   - Examples and Encodings
   - History Verification

3. Parameters and Quantification
   - Verifying Quantified Policies
1. Modelling behavioural information
   - Event Structures as a general model

2. A Simple Policy Language
   - Examples and Encodings
   - History Verification

3. Parameters and Quantification
   - Verifying Quantified Policies
An Event Structure model

**Protocols**
- Entities in a distributed system interact following protocols.
- Information about another entity is information about a number of (past) protocol runs with that entity.

**Event Structure Model of Information**
- A protocol can be specified as a concurrent process.
- Event structures were invented to give formal semantics to truly concurrent processes.
A model for behavioural information
Event Structures

- $ES = (E, \leq, \#)$, $E$ a set of events, $\leq$ and $\#$ relations on $E$.
- Information about a session is a finite set of events $x \subseteq E$, called a configuration (which is always conflict free and causally closed).
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in C_{ES}^*$, called a history.

EBay:

```
confirm   time-out
\downarrow
pay       ignore
\downarrow
positive  neutral  negative
```

e.g., $h = \{\text{pay, confirm, pos}\}{\text{pay, confirm, neu}}{\text{pay}}$

Karl Krukow (University of Aarhus, DK)  Concrete Reputation-Systems  πλSeminar, Daimi 8 / 22
A model for behavioural information
Event Structures

- $ES = (E, \leq, \#)$, $E$ a set of events, $\leq$ and $\#$ relations on $E$.
- Information about a session is a finite set of events $x \subseteq E$, called a configuration (which is always conflict free and causally closed).
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in C^*_E$, called a history.

**EBay:**

```
confirm  ~~~~ time-out
      ↖          ↖          ↗        ↗
    pay ~~~~~~~~~~ ignore
      ↙        ↙        ↙          ↙
positive ~~~~ neutral ~ negative
```

e.g., $h = \{\text{pay, confirm, pos}\} \{\text{pay, confirm, neu}\} \{\text{pay}\}$
A model for behavioural information
Event Structures

- \( ES = (E, \leq, \#) \), \( E \) a set of events, \( \leq \) and \( \# \) relations on \( E \).
- Information about a session is a finite set of events \( x \subseteq E \), called a configuration (which is always conflict free and causally closed).
- Information about several interactions is a sequence \( h = x_1 x_2 \cdots x_n \in C^*_{ES} \), called a history.

EBay:

\[
\begin{align*}
\text{confirm} & \sim \sim \sim \text{time-out} \\
\text{pay} & \sim \sim \sim \sim \sim \sim \text{ignore}
\end{align*}
\]

\[
\text{positive} \sim \sim \text{neutral} \sim \text{negative}
\]

E.g., \( h = \{\text{pay, confirm, pos}\}\{\text{pay, confirm, neu}\}\{\text{pay}\} \)
Some Problems
or choices at least...
Reputation System Security

If entity $p$ gains access to resource $r$ at time $t$, then the past behaviour of $p$ up until time $t$ satisfies requirement $\psi_r$.

- **Specification problem**: How to specify requirements $\psi_r$?
  - declaratively, expressively

- (Dynamic) **Verification problem**: given $h$ and $\psi_r$ does $h \models \psi_r$?
  - but information is provided incrementally
    - how to support operations: $h$.update($e$, $i$) and $h$.new().
  - so that given the “representation” of $h \models \psi_r$, the question $h$.op(…) $\models \psi_r$ should be efficient to answer.
Some Problems
or choices at least...

Reputation System Security

If entity $p$ gains access to resource $r$ at time $t$, then the past behaviour of $p$ up until time $t$ satisfies requirement $\psi_r$.

- **Specification problem**: How to specify requirements $\psi_r$?
  - declaratively, expressively
- **(Dynamic) Verification problem**: given $h$ and $\psi_r$ does $h \models \psi_r$?
  - but information is provided incrementally
    - how to support operations: $h.update(e, i)$ and $h.new()$.
  - so that given the “representation” of $h \models \psi_r$, the question $h.op(\ldots) \models \psi_r$ should be efficient to answer.
Outline

1. Modelling behavioural information
   - Event Structures as a general model

2. A Simple Policy Language
   - Examples and Encodings
   - History Verification

3. Parameters and Quantification
   - Verifying Quantified Policies
Pure-Past Linear Temporal Logic

- Syntax

\[ \psi ::= e \mid \Diamond e \mid \psi_0 \land \psi_1 \mid \psi_0 \lor \psi_1 \mid \neg \psi \mid X^{-1}\psi \mid \psi_0 S \psi_1 \]

- Semantics: relation \( \models \) between histories \( h = x_1 x_2 \cdots x_n \) and formulas \( \psi \).

\[
\begin{align*}
(h, i) \models e & \quad \text{iff} \quad e \in x_i \\
(h, i) \models \Diamond e & \quad \text{iff} \quad e \not\in x_i \\
(h, i) \models \psi_0 \land \psi_1 & \quad \text{iff} \quad (h, i) \models \psi_0 \text{ and } (h, i) \models \psi_1 \\
(h, i) \models \psi_0 \lor \psi_1 & \quad \text{iff} \quad (h, i) \models \psi_0 \text{ or } (h, i) \models \psi_1 \\
(h, i) \models \neg \psi & \quad \text{iff} \quad (h, i) \not\models \psi \\
(h, i) \models X^{-1}\psi & \quad \text{iff} \quad i > 0 \text{ and } (h, i - 1) \models \psi \\
(h, i) \models \psi_0 S \psi_1 & \quad \text{iff} \quad \exists j \leq i. (h, j) \models \psi_1 \text{ and } \\
& \quad \forall j'. j < j' \leq i \Rightarrow (h, j') \models \psi_0
\end{align*}
\]
Pure-Past Linear Temporal Logic

- **Syntax**

\[
\psi ::= e \mid \Diamond e \mid \psi_0 \land \psi_1 \mid \psi_0 \lor \psi_1 \mid \neg \psi \mid X^{-1} \psi \mid \psi_0 \mathcal{S} \psi_1
\]

- **Semantics**: relation \( \models \) between histories \( h = x_1 x_2 \cdots x_n \) and formulas \( \psi \).

\[
(h, i) \models e \quad \text{iff} \quad e \in x_i
\]

\[
(h, i) \models \Diamond e \quad \text{iff} \quad e \notin x_i
\]

\[
(h, i) \models \psi_0 \land \psi_1 \quad \text{iff} \quad (h, i) \models \psi_0 \text{ and } (h, i) \models \psi_1
\]

\[
(h, i) \models \psi_0 \lor \psi_1 \quad \text{iff} \quad (h, i) \models \psi_0 \text{ or } (h, i) \models \psi_1
\]

\[
(h, i) \models \neg \psi \quad \text{iff} \quad (h, i) \nmid \psi
\]

\[
(h, i) \models X^{-1} \psi \quad \text{iff} \quad i > 0 \text{ and } (h, i - 1) \models \psi
\]

\[
(h, i) \models \psi_0 \mathcal{S} \psi_1 \quad \text{iff} \quad \exists j \leq i. (h, j) \models \psi_1 \text{ and } \forall j'. j < j' \leq i \Rightarrow (h, j') \models \psi_0
\]

- \( h \models \psi \iff (h, \|h\|) \models \psi \quad (h \neq \epsilon) \)
A Simple Example

- **EBay Auction**
  - Policy: “only bid on auctions run by a seller that has never failed to send goods for won auctions in the past.”
    
    \[ \psi^{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \]
    
  - Furthermore, the buyer might require that “the seller has never provided negative feedback in auctions where payment was made.”
    
    \[ \psi^{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \land G^{-1}(\text{negative } \rightarrow \text{ ignore}) \]

- **History-Based Access Control?**
  - Can encode several policies from the literature.
  - “allow a program to connect to a remote site if and only if it has neither tried to open a local file it has not created, nor tried to modify a file it has created, nor tried to create a sub-process”?
A Simple Example

- **EBay Auction**
  - Policy: “only bid on auctions run by a seller that has never failed to send goods for won auctions in the past.”
    \[ \psi^{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \]
  - Furthermore, the buyer might require that “the seller has never provided negative feedback in auctions where payment was made.”
    \[ \psi^{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \land G^{-1}(\text{negative} \rightarrow \text{ignore}) \]

- **History-Based Access Control?**
  - Can encode several policies from the literature.
  - “allow a program to connect to a remote site if and only if it has neither tried to open a local file it has not created, nor tried to modify a file it has created, nor tried to create a sub-process”?

Karl Krukow (University of Aarhus, DK)
A Simple Example

- EBay Auction
  - Policy: “only bid on auctions run by a seller that has never failed to send goods for won auctions in the past.”
    \[ \psi_{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \]
  - Furthermore, the buyer might require that “the seller has never provided negative feedback in auctions where payment was made.”
    \[ \psi_{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \land G^{-1}(\text{negative} \rightarrow \text{ignore}) \]

- History-Based Access Control?
  - Can encode several policies from the litterature.
  - “allow a program to connect to a remote site if and only if it has neither tried to open a local file it has not created, nor tried to modify a file it has created, nor tried to create a sub-process”? 
A Simple Example

- **EBay Auction**
  - Policy: “only bid on auctions run by a seller that has never failed to send goods for won auctions in the past.”
    \[ \psi_{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \]
  - Furthermore, the buyer might require that “the seller has never provided negative feedback in auctions where payment was made.”
    \[ \psi_{\text{bid}} \equiv \neg F^{-1}(\text{time-out}) \land G^{-1}(\text{negative} \rightarrow \text{ignore}) \]

- **History-Based Access Control?**
  - Can encode several policies from the literature.
  - “allow a program to connect to a remote site if and only if it has neither tried to open a local file it has not created, nor tried to modify a file it has created, nor tried to create a sub-process”?

Karl Krukowski (University of Aarhus, DK)
A data structure for (dynamic) verification

- Goal is answering “$h \models \psi$?”
- Give a datastructure $DS$, maintaining a history $h$, and supporting three operations.
  - $DS$.new()
    $$(h \mapsto h\emptyset)$$
  - $DS$.update($e$, $i$)
    $$(h \mapsto h[i/(x_i \cup \{e\})])$$
  - $DS$.check()
    $$(h \models \psi?)$$
- Enumerate subformulas $\psi = \psi_0, \psi_1, \ldots, \psi_n$ (subformulas have higher indices).
Array-based Algorithm

Maintain

history \( h = x_1 \cdots x_n \), and boolean arrays \( B_1, \ldots, B_n \).

Invariant

\[(h, k) \models \psi_i \iff B_k[i] = \text{true}\]

<table>
<thead>
<tr>
<th>( x_k )</th>
<th>( x_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \top ) ( \psi_0 )</td>
<td>? ( \psi_0 )</td>
</tr>
<tr>
<td>( \bot ) ( \psi_1 )</td>
<td>? ( \psi_1 )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \bot ) ( \psi_i )</td>
<td>? ( \psi_i )</td>
</tr>
<tr>
<td>( \top ) ( \psi_{i+1} )</td>
<td>( \bot ) ( \psi_{i+2} )</td>
</tr>
</tbody>
</table>

Algorithm - case S

suppose \( \psi_i = \psi_{i+1} \lor \psi_{i+2} \)

then we can define

\[
B_{k+1}[i] = B_{k+1}[i + 2] \lor (B_k[i] \land B_{k+1}[i + 1])
\]

so we can fill array \( B_{k+1} \) in linear time (in \(|\psi|\)) given \( B_k \).
Array-based Algorithm

Maintain

history $h = x_1 \cdots x_n$, and boolean arrays $B_1, \ldots, B_n$.

Invariant

$(h, k) \models \psi_i \iff B_k[i] = \text{true}$

Algorithm - case S

suppose $\psi_i = \psi_{i+1} \land \psi_{i+2}$

then we can define

$$B_{k+1}[i] = B_{k+1}[i+2] \lor (B_k[i] \land B_{k+1}[i+1])$$

so we can fill array $B_{k+1}$ in linear time (in $|\psi|$) given $B_k$. 
Array-based Algorithm

Maintain

history \( h = x_1 \cdots x_n \), and boolean arrays \( B_1, \ldots, B_n \).

Invariant

\((h, k) \models \psi_i \iff B_k[i] = \text{true}\)

Algorithm - case S

suppose \( \psi_i = \psi_{i+1} \ S \psi_{i+2} \) then we can define

\[
B_{k+1}[i] = B_{k+1}[i + 2] \lor (B_k[i] \land B_{k+1}[i + 1])
\]

so we can fill array \( B_{k+1} \) in linear time (in \(|\psi|\)) given \( B_k \).
### Array-based Algorithm

**Maintain**

history \( h = x_1 \cdots x_n \), and boolean arrays \( B_1, \ldots, B_n \).

<table>
<thead>
<tr>
<th>( x_k )</th>
<th>( x_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>( \perp ) ( \psi_i )</td>
<td>( \perp ) ( \psi_i )</td>
</tr>
<tr>
<td>( \perp ) ( \psi_i+1 )</td>
<td>( \perp ) ( \psi_i+1 )</td>
</tr>
<tr>
<td>( \perp ) ( \psi_i+2 )</td>
<td>( \perp ) ( \psi_i+2 )</td>
</tr>
</tbody>
</table>

**Invariant**

\[(h, k) \models \psi_i \iff B_k[i] = \text{true}\]

**Algorithm - case S**

suppose \( \psi_i = \psi_{i+1} \mathbin{S} \psi_{i+2} \)

then we can define

\[
B_{k+1}[i] = B_{k+1}[i + 2] \lor (B_k[i] \land B_{k+1}[i + 1])
\]

so we can fill array \( B_{k+1} \) in linear time (in \(|\psi|\)) given \( B_k \).
An automata-based algorithm

- Consider $x_1 x_2 \cdots x_n \models \psi$? as an acceptance problem for an automata reading symbols from $C_{ES}$.
- Language $L_\psi = \{ h \in C_{ES}^* \mid h \models \psi \}$ is regular.
  - Transition $s \xrightarrow{x_i} s'$ depends only current state $s$ and configuration $x_i$.
  - Minimization.
  - Precomputation of transitions (factor $|\psi|$ at runtime).
Outline

1. Modelling behavioural information
   - Event Structures as a general model

2. A Simple Policy Language
   - Examples and Encodings
   - History Verification

3. Parameters and Quantification
   - Verifying Quantified Policies
Parameters and Quantification

- Recall example property: “...[never] open a local file that it has not created...”
- want for any file f “if open(f) then F⁻¹ create(f).”
  - but infinitely many possible filenames
- Need a notion of parameterized event structure.
  - events e occur with parameters p from (infinite) parameter sets P
  - otherwise as usual event structures
- Specify property as

  \[ G^{-1} \left( \forall x. \left[ \text{open}(x) \rightarrow F^{-1}(\text{create}(x)) \right] \right) \]
Extended Policy Language

- Extended language $\psi ::= \cdots e(v) \mid \cdots | Qx : P_i.\psi$
  ($v$ ranges over variables and constant parameters)
- Histories $h$ are now sequences of configurations from parameterized event-structures.
  - A configuration $x_i$ partially maps events to parameters.
- Semantics is now relative to an environment $\sigma$. E.g.,

  $$(h, i) \models^\sigma e(v) \iff e \in dom(x_i) \text{ and } x_i(e) = \sigma(v)$$

  $$(h, i) \models^\sigma \forall x : P_j.\psi \iff \forall p \in P_j.(h, i) \models^{(x \mapsto p)/\sigma} \psi$$
Constraints

- Verification problem: Given history $h$ and quantified policy $\psi'$, does $h \models \psi'$?
- We can generalize boolean array algorithm.
- Notion of a constraint

$$c ::= \bot | (x = p) | c \land c | c \lor c | \neg c \quad (x \in \text{Var}, p \text{ is a parameter})$$

- A propositional logic.
- Semantics: $\sigma \models c$ between environments $\sigma$ and constraints $c$.
- We can map $(h, k, \psi)$ into a constraint $\llbracket \psi \rrbracket_h^k$, e.g.,

$$\llbracket e(x) \rrbracket_h^k = (x = p) \text{ if } h^k(e) = p.$$
Constraint-Array Algorithm

Maintain

history \( h = x_1 \cdots x_n \), and boolean arrays \( B_1, \ldots, B_n \).

Invariant

\[(h, k) \models \psi_i \iff B_k[i] = \text{true}\]

Algorithm - case S

suppose \( \psi_i = \psi_{i+1} S \psi_{i+2} \)

then we can define

\[C_{k+1}[i] = C_{k+1}[i + 2] \lor (C_k[i] \land C_{k+1}[i + 1])\]

so we can fill array \( C_{k+1} \) in linear time (in \(|\psi|\)) given \( C_k \).
Constraint-Array Algorithm

Maintain

history \( h = x_1 \cdots x_n \), and constraint arrays \( C_1, \ldots, C_n \)

\[
C_k \text{ is defined as: } \begin{cases} \top & \text{if } \psi_i \in h \\ \bot & \text{otherwise} \end{cases}
\]

Invariant

\[
\forall \sigma. \left( (h, k) \models \psi_i \iff \sigma \models C_k[i] \right)
\]

Algorithm - case S

suppose \( \psi_i = \psi_{i+1} \) S \( \psi_{i+2} \)

then we can define

\[
C_{k+1}[i] = C_{k+1}[i + 2] \lor (C_k[i] \land C_{k+1}[i + 1])
\]

so we can fill array \( C_{k+1} \) in linear time (in \( |\psi| \)) given \( C_k \).
Constraint-Array Algorithm

Maintain

history \( h = x_1 \cdots x_n \), and
constraint arrays \( C_1, \ldots, C_n \)

Invariant

\[ \forall \sigma. [(h, k) \models^\sigma \psi_i \iff \sigma \models C_k[i]] \]

Algorithm - case S

suppose \( \psi_i = \psi_{i+1} \quad S \quad \psi_{i+2} \)
then we can define

\[ C_{k+1}[i] = C_{k+1}[i + 2] \lor (C_k[i] \land C_{k+1}[i + 1]) \]
so we can fill array \( C_{k+1} \) in linear time (in \(|\psi|\)) given \( C_k \).
Constraint-Array Algorithm

**Maintain**

- history $h = x_1 \cdots x_n$, and
- constraint arrays $C_1, \ldots, C_n$

**Invariant**

$\forall \sigma. [(h, k) \models^\sigma \psi_i \iff \sigma \models C_k[i]]$

**Algorithm - case S**

Suppose $\psi_i = \psi_{i+1} \text{ S } \psi_{i+2}$

Then we can define

$$C_{k+1}[i] = C_{k+1}[i + 2] \lor (C_k[i] \land C_{k+1}[i + 1])$$

So we can fill array $C_{k+1}$ in linear time (in $|\psi|$) given $C_k$. 
Constraint-Array Algorithm

Maintain

history \( h = x_1 \cdots x_n \), and
constraint arrays \( C_1, \ldots, C_n \)

\[
\begin{array}{c|c}
X_k & X_{k+1} \\
\hline
[\psi_0]^k_h & C \\
[\psi_1]^k_h & C \\
\vdots & \vdots \\
[\psi_i]^k_h & C \\
\end{array}
\]

\[
\begin{array}{c|c}
& [\psi_0]^{k+1}_h \\
\hline
? & ? \\
\vdots & \vdots \\
C & [\psi_i]^{k+1}_h \\
C & [\psi_{i+1}]^{k+1}_h \\
\end{array}
\]

Invariant

\( \forall \sigma. ([h, k] \models \sigma \psi_i \iff \sigma \models C_k[i]) \)

Algorithm - case \( S \)

suppose \( \psi_i = \psi_{i+1} S \psi_{i+2} \)
then we can define

\[
C_{k+1}[i] = C_{k+1}[i + 2] \lor (C_k[i] \land C_{k+1}[i + 1])
\]

so we can fill array \( C_{k+1} \) in linear time (in \( |\psi| \)) given \( C_k \).
Verifying Quantified Policies

- But how to eliminate quantifiers, e.g., $\forall x : P.\psi$
  - Suppose $c = (x = p \land y \neq q) \lor (x \neq p \land y = q')$.
  - We must now produce a constraint $c'$ (without $x$) so that
    
    $$
    \sigma \models c' \iff \left[ \forall p \in P_i.([x \mapsto p]/\sigma) \models c \right] \quad \text{(for all } \sigma) 
    $$
    
  - this becomes $y \neq q \land y = q'$
  - In general, we can eliminate a variable $x$ and obtain constraint equivalent to $\forall x$ or $\exists x$.

- Caveat: deciding $h \models \psi$ (for closed $\psi$) even in small models is $PSPACE$ complete.
  - (reduction from quantified boolean logic)
But how to eliminate quantifiers, e.g., $\forall x : P . \psi$

- Suppose $c = (x = p \land y \neq q) \lor (x \neq p \land y = q')$.
- We must now produce a constraint $c'$ (without $x$) so that

$$
\sigma \models c' \iff \forall p \in P_i.([x \mapsto p]/\sigma) \models c \quad \text{(for all } \sigma)$$

- this becomes $y \neq q \land y = q'$
- In general, we can eliminate a variable $x$ and obtain constraint equivalent to $\forall x$ or $\exists x$.

Caveat: deciding $h \models \psi$ (for closed $\psi$) even in small models is $PSPACE$ complete.

- (reduction from quantified boolean logic)
Summary

- A framework for “reputation systems” and a notion of “security” (or correctness) of these systems.
  - applications in history-based access control.
- Basic Policies can be declaratively specified and efficiently verified.
- Quantified policies are more expressive, and quantified model checking is decidable (though hard with many quantifiers).

Future Work?
- Tighten bound on quantified algorithm