A Framework for Concrete Reputation Systems

with applications to History-Based Access Control

Vladimiro Sassone

Department of Informatics
University of Sussex

7 July 2005

Access Control

Resource access control is paramount for open-ended systems:

Correctness

If entity p gets access to resource r, then p is "authorised" to access r.

Different mechanisms provide for different meanings of "authorised."

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

Access Control

Resource access control is paramount for open-ended systems:

Correctness

If entity p gets access to resource r, then p is "authorised" to access r.

Different mechanisms provide for different meanings of "authorised."

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

Access Control

Resource access control is paramount for open-ended systems:

Correctness

If entity p gets access to resource r, then p is "authorised" to access r.

Different mechanisms provide for different meanings of "authorised."

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

Access Control

Resource access control is paramount for open-ended systems:

Correctness

If entity p gets access to resource r, then p is “authorised” to access r.

Different mechanisms provide for different meanings of “authorised.”

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

Reputation Systems

and *dynamic* trust management...

Reputation

- Behaviour-based: an entity's (perceived) behaviour in past interactions is used to determine its privilege in future ones.
- Relevant for large decentralised systems with multiple interactions.

But, when is an entity in a reputation system "authorised"?

- Existing systems provide no "correctness" criteria.
- often "*reputation information*" undergoes heavy abstraction – e.g., Eigentrust and Ebay.

Reputation System Security

The degree of confidence (trust) in p's actions at time t, is determined by p's behaviour up *until* time t according to a given policy ψ.

Reputation Systems

and *dynamic* trust management…

Reputation

- Behaviour-based: an entity's (perceived) behaviour in past interactions is used to determine its privilege in future ones.
- Relevant for large decentralised systems with multiple interactions.

But, when is an entity in a reputation system "authorised"?

- Existing systems provide no "correctness" criteria.
- often "*reputation information*" undergoes heavy abstraction – e.g., Eigentrust and Ebay.

Reputation System Security

The degree of confidence (trust) in p's actions at time t, is determined by p's behaviour up *until* time t according to a given policy ψ.

Reputation Systems

and *dynamic* trust management. . .

Reputation

- Behaviour-based: an entity's (perceived) behaviour in past interactions is used to determine its privilege in future ones.
- Relevant for large decentralised systems with multiple interactions.

But, when is an entity in a reputation system "authorised"?

- Existing systems provide no "correctness" criteria.
- often "*reputation information*" undergoes heavy abstraction – e.g., Eigentrust and Ebay.

Reputation System Security

The degree of confidence (trust) in p's actions at time t, is determined by p's behaviour up *until* time t according to a given policy ψ.

History-Based Access Control

and reputation systems

Example:

- Suppose you download what claims to be a new cool browser from some unknown site. Your trust policy may be:
- *allow the 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 to modify a file it has created, nor to create a sub-process.*

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

Reputation-Based Access Control

If entity p gains access to resource r at time t, then p's behaviour up *until* time t satisfies a given requirement ψ_r.

History-Based Access Control
and reputation systems

Example:

- Suppose you download what claims to be a new cool browser from some unknown site. Your trust policy may be:
- *allow the 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 to modify a file it has created, nor to create a sub-process.*

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

Reputation-Based Access Control

If entity p gains access to resource r at time t, then p's behaviour up *until* time t satisfies a given requirement ψ_r.

Outline

Outline

A Model based on Event Structure

Interactions and Protocols

- At an abstract level, entities in a distributed system interact according to protocols;
- Information about an external entity is just information about a number of (past) protocol runs with that entity.

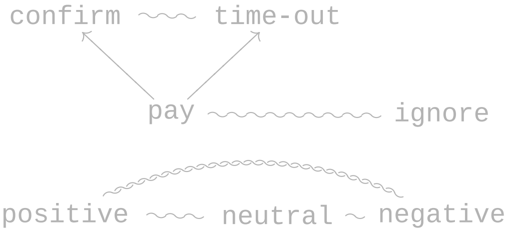
Events as Model of Information

- A protocol can be specified as a concurrent process, at different levels of abstractions.
- Event structures were invented to give formal semantics to truely concurrent processes, expressing "causation" and "conflict."

A model for behavioural information

- $ES = (E, \leq, \#)$, with 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 'conflict-free' and 'causally-closed').
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in \mathcal{C}^*_{ES}$, called a history.

EBay (simplified) example:

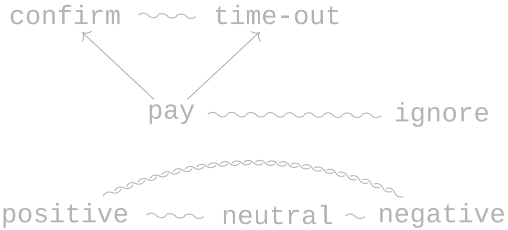


e.g., $h = \{\texttt{pay}, \texttt{confirm}, \texttt{pos}\}\ \{\texttt{pay}, \texttt{confirm}, \texttt{neu}\}\ \{\texttt{pay}\}$

A model for behavioural information

- $ES = (E, \leq, \#)$, with 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 'conflict-free' and 'causally-closed').
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in \mathcal{C}^*_{ES}$, called a history.

EBay (simplified) example:

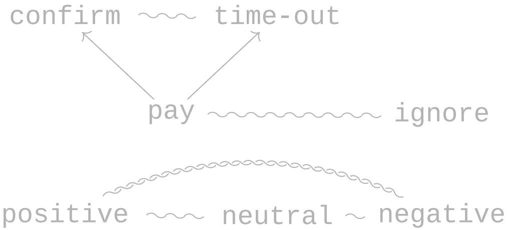


e.g., $h = \{\mathtt{pay}, \mathtt{confirm}, \mathtt{pos}\}\ \{\mathtt{pay}, \mathtt{confirm}, \mathtt{neu}\}\ \{\mathtt{pay}\}$

A model for behavioural information

- $ES = (E, \leq, \#)$, with 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 'conflict-free' and 'causally-closed').
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in \mathcal{C}^*_{ES}$, called a history.

EBay (simplified) example:

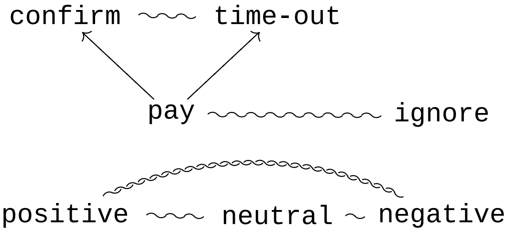


e.g., $h = \{\mathtt{pay}, \mathtt{confirm}, \mathtt{pos}\}\ \{\mathtt{pay}, \mathtt{confirm}, \mathtt{neu}\}\ \{\mathtt{pay}\}$

A model for behavioural information

- $ES = (E, \leq, \#)$, with 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 'conflict-free' and 'causally-closed').
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in \mathcal{C}^*_{ES}$, called a history.

EBay (simplified) example:

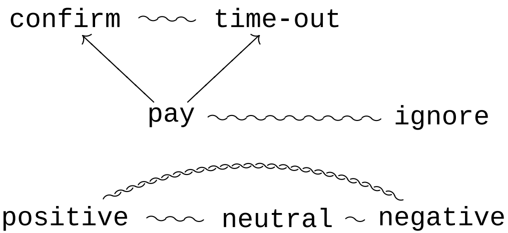


e.g., $h = \{\texttt{pay}, \texttt{confirm}, \texttt{pos}\}\ \{\texttt{pay}, \texttt{confirm}, \texttt{neu}\}\ \{\texttt{pay}\}$

A model for behavioural information

- $ES = (E, \leq, \#)$, with 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 'conflict-free' and 'causally-closed').
- Information about several interactions is a sequence $h = x_1 x_2 \cdots x_n \in \mathcal{C}_{ES}^*$, called a history.

EBay (simplified) example:



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

The central issues

Reputation System Security

If entity p gains access to resource r at time t, then the p's behaviour up *until* t satisfies requirement ψ_r.

Specification problem: How to specify requirements ψ_r?

- The language must be expressive, intuitive, declarative, ...

Verification problem: given h and ψ_r does $h \models \psi_r$?

- but information is provided incrementally: the model checking must be dynamic, i.e., support the operations $h.\mathbf{update}(e, i)$ and $h.\mathbf{new}()$.
- and, of course, the "representation" of h must be such that the question $h \models \psi_r$ is efficient to answer.

The central issues

Reputation System Security

If entity p gains access to resource r at time t, then the p's behaviour up *until* t satisfies requirement ψ_r.

Specification problem: How to specify requirements ψ_r?

- The language must be expressive, intuitive, declarative, ...

Verification problem: given h and ψ_r does $h \models \psi_r$?

- but information is provided incrementally: the model checking must be dynamic, i.e., support the operations $h.\mathbf{update}(e, i)$ and $h.\mathbf{new}()$.
- and, of course, the "representation" of h must be such that the question $h \models \psi_r$ is efficient to answer.

The central issues

Reputation System Security

If entity p gains access to resource r at time t, then the p's behaviour up *until* t satisfies requirement ψ_r.

Specification problem: How to specify requirements ψ_r?

- The language must be expressive, intuitive, declarative, ...

Verification problem: given h and ψ_r does $h \models \psi_r$?

- but information is provided incrementally: the model checking must be dynamic, i.e., support the operations $h.\mathbf{update}(e, i)$ and $h.\mathbf{new}()$.
- and, of course, the "representation" of h must be such that the question $h \models \psi_r$ is efficient to answer.

Outline

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 | \psi_0 \vee \psi_1 \mid \neg\psi \mid X^{-1}\psi \mid \psi_0 \, S \, \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad (h \neq \epsilon)$$

$$\begin{array}{lll}
(h,i) \models e & \text{iff} & e \in x_i \\
(h,i) \models \Diamond e & \text{iff} & e \,\#\, x_i \\
(h,i) \models \psi_0 \wedge \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ and } (h,i) \models \psi_1 \\
(h,i) \models \psi_0 \vee \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ or } (h,i) \models \psi_1 \\
(h,i) \models \neg\psi & \text{iff} & (h,i) \not\models \psi \\
(h,i) \models X^{-1}\psi & \text{iff} & i > 0 \text{ and } (h,i-1) \models \psi \\
(h,i) \models \psi_0 \, S \, \psi_1 & \text{iff} & \exists j \leq i.(h,j) \models \psi_1 \text{ and} \\
 & & \quad \forall j'.j < j' \leq i \Rightarrow (h,j') \models \psi_0
\end{array}$$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 \mid \psi_0 \vee \psi_1 \mid \neg\psi \mid \mathsf{X}^{-1}\psi \mid \psi_0 \; \mathsf{S} \; \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad\qquad (h \neq \epsilon)$$

$$\begin{array}{lll}
(h,i) \models e & \text{iff} & e \in x_i \\
(h,i) \models \Diamond e & \text{iff} & e \;\#\; x_i \\
(h,i) \models \psi_0 \wedge \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ and } (h,i) \models \psi_1 \\
(h,i) \models \psi_0 \vee \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ or } (h,i) \models \psi_1 \\
(h,i) \models \neg\psi & \text{iff} & (h,i) \not\models \psi \\
(h,i) \models \mathsf{X}^{-1}\psi & \text{iff} & i > 0 \text{ and } (h, i-1) \models \psi \\
(h,i) \models \psi_0 \; \mathsf{S} \; \psi_1 & \text{iff} & \exists j \leq i. (h,j) \models \psi_1 \text{ and} \\
 & & \qquad \forall j'. j < j' \leq i \Rightarrow (h,j') \models \psi_0
\end{array}$$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 | \psi_0 \vee \psi_1 \mid \neg\psi \mid \mathsf{X}^{-1}\psi \mid \psi_0 \; \mathsf{S} \; \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad (h \neq \epsilon)$$

$(h,i) \models e$	iff	$e \in x_i$
$(h,i) \models \Diamond e$	iff	$e \; \# \; x_i$
$(h,i) \models \psi_0 \wedge \psi_1$	iff	$(h,i) \models \psi_0$ and $(h,i) \models \psi_1$
$(h,i) \models \psi_0 \vee \psi_1$	iff	$(h,i) \models \psi_0$ or $(h,i) \models \psi_1$
$(h,i) \models \neg\psi$	iff	$(h,i) \not\models \psi$
$(h,i) \models \mathsf{X}^{-1}\psi$	iff	$i > 0$ and $(h, i-1) \models \psi$
$(h,i) \models \psi_0 \; \mathsf{S} \; \psi_1$	iff	$\exists j \leq i.(h,j) \models \psi_1$ and $\forall j'.j < j' \leq i \Rightarrow (h,j') \models \psi_0$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 | \psi_0 \vee \psi_1 \mid \neg\psi \mid \mathsf{X}^{-1}\psi \mid \psi_0 \; \mathsf{S} \; \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad\qquad (h \neq \epsilon)$$

$$
\begin{array}{lll}
(h,i) \models e & \text{iff} & e \in x_i \\
(h,i) \models \Diamond e & \text{iff} & e \,\#\, x_i \\
(h,i) \models \psi_0 \wedge \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ and } (h,i) \models \psi_1 \\
(h,i) \models \psi_0 \vee \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ or } (h,i) \models \psi_1 \\
(h,i) \models \neg\psi & \text{iff} & (h,i) \not\models \psi \\
(h,i) \models \mathsf{X}^{-1}\psi & \text{iff} & i > 0 \text{ and } (h, i-1) \models \psi \\
(h,i) \models \psi_0 \; \mathsf{S} \; \psi_1 & \text{iff} & \exists j \leq i.(h,j) \models \psi_1 \text{ and} \\
 & & \qquad \forall j'.j < j' \leq i \Rightarrow (h,j') \models \psi_0
\end{array}
$$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 \mid \psi_0 \vee \psi_1 \mid \neg\psi \mid X^{-1}\psi \mid \psi_0 \, S \, \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad\qquad (h \neq \epsilon)$$

$$
\begin{array}{lll}
(h,i) \models e & \text{iff} & e \in x_i \\
(h,i) \models \Diamond e & \text{iff} & e \ \# \ x_i \\
(h,i) \models \psi_0 \wedge \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ and } (h,i) \models \psi_1 \\
(h,i) \models \psi_0 \vee \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ or } (h,i) \models \psi_1 \\
(h,i) \models \neg\psi & \text{iff} & (h,i) \not\models \psi \\
(h,i) \models X^{-1}\psi & \text{iff} & i > 0 \text{ and } (h, i-1) \models \psi \\
(h,i) \models \psi_0 \, S \, \psi_1 & \text{iff} & \exists j \leq i. (h,j) \models \psi_1 \text{ and} \\
 & & \qquad \forall j'. j < j' \leq i \Rightarrow (h,j') \models \psi_0
\end{array}
$$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 \mid \psi_0 \vee \psi_1 \mid \neg\psi \mid \mathsf{X}^{-1}\psi \mid \psi_0 \; \mathsf{S} \; \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad (h \neq \epsilon)$$

$(h,i) \models e$	iff	$e \in x_i$
$(h,i) \models \Diamond e$	iff	$e \;\#\; x_i$
$(h,i) \models \psi_0 \wedge \psi_1$	iff	$(h,i) \models \psi_0$ and $(h,i) \models \psi_1$
$(h,i) \models \psi_0 \vee \psi_1$	iff	$(h,i) \models \psi_0$ or $(h,i) \models \psi_1$
$(h,i) \models \neg\psi$	iff	$(h,i) \not\models \psi$
$(h,i) \models \mathsf{X}^{-1}\psi$	iff	$i > 0$ and $(h, i-1) \models \psi$
$(h,i) \models \psi_0 \; \mathsf{S} \; \psi_1$	iff	$\exists j \leq i.(h,j) \models \psi_1$ and $\forall j'.j < j' \leq i \Rightarrow (h,j') \models \psi_0$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 | \psi_0 \vee \psi_1 \mid \neg\psi \mid \mathsf{X}^{-1}\psi \mid \psi_0 \; \mathsf{S} \; \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad\qquad (h \neq \epsilon)$$

$$
\begin{array}{lll}
(h,i) \models e & \text{iff} & e \in x_i \\
(h,i) \models \Diamond e & \text{iff} & e \;\#\; x_i \\
(h,i) \models \psi_0 \wedge \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ and } (h,i) \models \psi_1 \\
(h,i) \models \psi_0 \vee \psi_1 & \text{iff} & (h,i) \models \psi_0 \text{ or } (h,i) \models \psi_1 \\
(h,i) \models \neg\psi & \text{iff} & (h,i) \not\models \psi \\
(h,i) \models \mathsf{X}^{-1}\psi & \text{iff} & i > 0 \text{ and } (h, i-1) \models \psi \\
(h,i) \models \psi_0 \; \mathsf{S} \; \psi_1 & \text{iff} & \exists j \leq i.(h,j) \models \psi_1 \text{ and} \\
 & & \qquad \forall j'.j < j' \leq i \Rightarrow (h,j') \models \psi_0
\end{array}
$$

Pure-Past Linear Temporal Logic

- Syntax

$$\psi \quad ::= \quad e \mid \Diamond e \mid \psi_0 \wedge \psi_1 \mid \psi_0 \vee \psi_1 \mid \neg\psi \mid \mathsf{X}^{-1}\psi \mid \psi_0 \; \mathsf{S} \; \psi_1$$

- Semantics: forcing $\models$ of formulas ψ by histories $h = x_1 x_2 \cdots x_n$

$$h \models \psi \iff (h, |h|) \models \psi \qquad (h \neq \epsilon)$$

$(h,i) \models e$	iff	$e \in x_i$
$(h,i) \models \Diamond e$	iff	$e \; \# \; x_i$
$(h,i) \models \psi_0 \wedge \psi_1$	iff	$(h,i) \models \psi_0$ and $(h,i) \models \psi_1$
$(h,i) \models \psi_0 \vee \psi_1$	iff	$(h,i) \models \psi_0$ or $(h,i) \models \psi_1$
$(h,i) \models \neg\psi$	iff	$(h,i) \not\models \psi$
$(h,i) \models \mathsf{X}^{-1}\psi$	iff	$i > 0$ and $(h, i-1) \models \psi$
$(h,i) \models \psi_0 \; \mathsf{S} \; \psi_1$	iff	$\exists j \leq i.(h,j) \models \psi_1$ and $\forall j'.j < j' \leq i \Rightarrow (h,j') \models \psi_0$

A Simple Example

eBay Auction

- Policy: *"only bids on auctions run by a seller that has never failed to send goods for won auctions in the past."*

$$\psi^{\text{bid}} \equiv \neg \mathsf{F}^{-1}(\texttt{time-out})$$

- Furthermore, the buyer might require that *"the seller has never provided negative feedback in auctions where payment was made."*

$$\psi^{\text{bid}} \equiv \neg \mathsf{F}^{-1}(\texttt{time-out}) \wedge \mathsf{G}^{-1}(\texttt{negative} \rightarrow \texttt{ignore})$$

A Simple Example

eBay Auction

- Policy: *"only bids on auctions run by a seller that has never failed to send goods for won auctions in the past."*

$$\psi^{\mathsf{bid}} \equiv \neg \mathsf{F}^{-1}(\texttt{time-out})$$

- Furthermore, the buyer might require that *"the seller has never provided negative feedback in auctions where payment was made."*

$$\psi^{\mathsf{bid}} \equiv \neg \mathsf{F}^{-1}(\texttt{time-out}) \wedge \mathsf{G}^{-1}(\texttt{negative} \rightarrow \texttt{ignore})$$

An efficient algorithm for (dynamic) verification

Goal: To answer " $h \models \psi$?"

- Identify a datastructure *DMC*, maintaining a history h, and supporting three methods:
  - *DMC*.**new**() $\quad h \mapsto h\emptyset$
  - *DMC*.**update**(e, i) $\quad h \mapsto h[i/(x_i \cup \{e\})]$
  - *DMC*.**check**() $\quad h \models \psi?$

In the following fix an enumeration of subformulas of ψ:

$$\psi_0 = \psi_1 \wedge \psi_2 = \neg \mathsf{F}^{-1}(\texttt{time-out}) \wedge \mathsf{G}^{-1}(\texttt{negative} \rightarrow \texttt{ignore})$$

$$\psi_1 = \neg \psi_3$$

$$\psi_2 = \mathsf{G}^{-1}(\psi_4)$$

$$\psi_3 = \mathsf{F}^{-1}(\psi_5)$$

$$\psi_4 = \psi_6 \rightarrow \psi_7$$

$$\psi_5 = \texttt{time-out}$$

$$\psi_6 = \texttt{negative}$$

$$\psi_7 = \texttt{ignore}$$

An efficient algorithm for (dynamic) verification

Goal: To answer " $h \models \psi$?"

- Identify a datastructure *DMC*, maintaining a history *h*, and supporting three methods:
  - *DMC*.**new**() $\quad h \mapsto h\emptyset$
  - *DMC*.**update**(e, i) $\quad h \mapsto h[i/(x_i \cup \{e\})]$
  - *DMC*.**check**() $\quad h \models \psi$?

In the following fix an enumeration of subformulas of ψ:

$$\psi_0 = \psi_1 \wedge \psi_2 \qquad = \neg \mathsf{F}^{-1}(\texttt{time-out}) \wedge \mathsf{G}^{-1}(\texttt{negative} \rightarrow \texttt{ignore})$$

$$\psi_1 = \neg \psi_3$$

$$\psi_2 = \mathsf{G}^{-1}(\psi_4)$$

$$\psi_3 = \mathsf{F}^{-1}(\psi_5)$$

$$\psi_4 = \psi_6 \rightarrow \psi_7$$

$$\psi_5 = \texttt{time-out}$$

$$\psi_6 = \texttt{negative}$$

$$\psi_7 = \texttt{ignore}$$

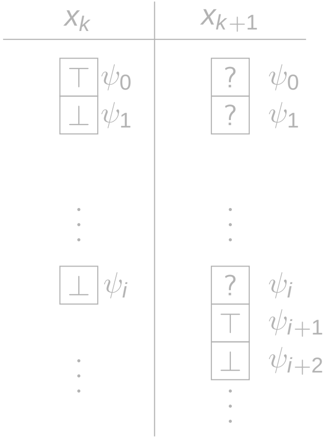
Array-based Algorithm

Maintain

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

Invariant

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



Algorithm – case S

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

$$B_{k+1}[i] \;=\; B_{k+1}[i+2] \vee (B_k[i] \wedge B_{k+1}[i+1])$$

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

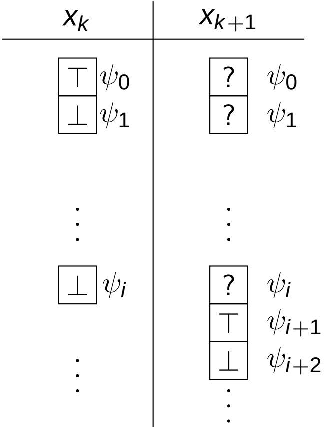
Array-based Algorithm

Maintain

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

Invariant

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



Algorithm – case S

suppose $\psi_i = \psi_{i+1} \mathrel{\mathsf{S}} \psi_{i+2}$
then we can define

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

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

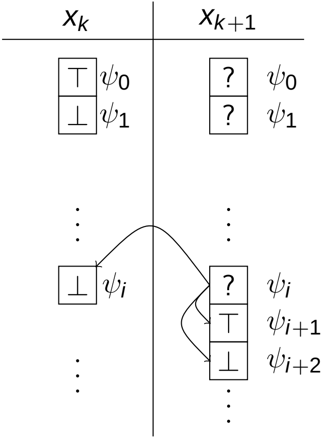
Array-based Algorithm

Maintain

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

Invariant

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



Algorithm – case S

suppose $\psi_i = \psi_{i+1} \,\mathsf{S}\, \psi_{i+2}$
then we can define

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

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

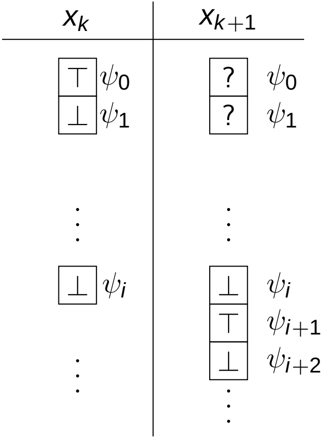
Array-based Algorithm

Maintain

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

Invariant

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



Algorithm – case S

suppose $\psi_i = \psi_{i+1} \mathrel{\mathsf{S}} \psi_{i+2}$
then we can define

$$B_{k+1}[i] \;=\; B_{k+1}[i+2] \vee (B_k[i] \wedge B_{k+1}[i+1])$$

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

Complexity results

Theorem

$DMC.\textbf{init}()$	$O(\|\psi\|)$
$DMC.\textbf{new}()$	$O(\|\psi\|)$
$DMC.\textbf{update}(e, i)$	$O((K - i + 1) \cdot \|\psi\|)$
$DMC.\textbf{check}()$	$O(1)$

An automata-based algorithm

Consider $x_1 x_2 \cdots x_n \models \psi$? as an acceptance problem for an automata reading symbols from $\mathcal{C}_{ES}$.

Theorem

Language $L_\psi = \{h \in {\mathcal{C}_{ES}}^* \mid h \models \psi\}$ is regular. Can identify an automata to recognise the "good" histories.

Transition $s \xrightarrow{x_i} s'$ depends only on current state s and configuration x_i.

Complexity: In fact, this amounts to precompute the transitions, and save a factor $|\psi|$ at runtime at the price of a cost at startup time.

DMC.**init**()	$O(2^{|\psi|} \cdot |\mathcal{C}_{ES}| \cdot |\psi|)$
DMC.**new**()	$O(1)$
DMC.**update**(e, i)	$O(K - i + 1)$
DMC.**check**()	$O(1)$

An automata-based algorithm

Consider $x_1 x_2 \cdots x_n \models \psi$? as an acceptance problem for an automata reading symbols from $\mathcal{C}_{ES}$.

Theorem

Language $L_\psi = \{h \in {\mathcal{C}_{ES}}^* \mid h \models \psi\}$ is regular. Can identify an automata to recognise the "good" histories.

Transition $s \xrightarrow{x_i} s'$ depends only on current state s and configuration x_i.

Complexity: In fact, this amounts to precompute the transitions, and save a factor $|\psi|$ at runtime at the price of a cost at startup time.

DMC.**init**()	$O(2^{|\psi|} \cdot |\mathcal{C}_{ES}| \cdot |\psi|)$
DMC.**new**()	$O(1)$
DMC.**update**(e, i)	$O(K - i + 1)$
DMC.**check**()	$O(1)$

An automata-based algorithm

Consider $x_1 x_2 \cdots x_n \models \psi$? as an acceptance problem for an automata reading symbols from $\mathcal{C}_{ES}$.

Theorem

Language $L_\psi = \{h \in {\mathcal{C}_{ES}}^* \mid h \models \psi\}$ *is regular. Can identify an automata to recognise the "good" histories.*

Transition $s \xrightarrow{x_i} s'$ depends only on current state s and configuration x_i.

Complexity: In fact, this amounts to precompute the transitions, and save a factor $|\psi|$ at runtime at the price of a cost at startup time.

$DMC.\mathbf{init}()$	$O(2^{|\psi|} \cdot |\mathcal{C}_{ES}| \cdot |\psi|)$
$DMC.\mathbf{new}()$	$O(1)$
$DMC.\mathbf{update}(e, i)$	$O(K - i + 1)$
$DMC.\mathbf{check}()$	$O(1)$

Outline

Parameters and Quantification

Recall example property:

"...[never] open a local file that it has not created ..."

Want *for any* file f:

"if $\texttt{open}(f)$ then $\mathsf{F}^{-1}\texttt{create}(f)$"

Need a notion of *parametrised* event structure.

- events e occur with parameters p from (infinite) parameter sets P
- otherwise as usual event structures

Specify property as

$$\mathsf{G}^{-1}\left(\forall x.\left[\texttt{open}(x) \rightarrow \mathsf{F}^{-1}(\texttt{create}(x))\right]\right)$$

Extended Policy Language

$$\psi ::= \cdots e(v) \mid \diamond e(v) \mid \cdots \mid Qx : P.\psi$$

v is a variable or a parameter, P is a parameter set, Q is $\forall$ or $\exists$.

- Histories h are now sequences of configurations from parameterised event-structures.
- A configuration x_i is a partial map events $\rightharpoonup$ parameters.

Semantics is relative to an environment σ:

$$
\begin{array}{lll}
(h,i) \models^{\sigma} e(v) & \text{iff} & e \in dom(x_i) \text{ and } x_i(e) = \sigma(v) \\
 & \vdots & \\
(h,i) \models^{\sigma} \forall x : P_j.\psi & \text{iff} & \text{for all } p \in P_j.(h,i) \models^{((x \mapsto p)/\sigma)} \psi \\
(h,i) \models^{\sigma} \exists x : P_j.\psi & \text{iff} & \text{there exists } p \in P_j.(h,i) \models^{((x \mapsto p)/\sigma)} \psi
\end{array}
$$

Extended Policy Language

$$\psi ::= \cdots e(v) \mid \Diamond e(v) \mid \cdots \mid Qx : P.\psi$$

v is a variable or a parameter, P is a parameter set, Q is $\forall$ or $\exists$.

- Histories h are now sequences of configurations from parameterised event-structures.
- A configuration x_i is a partial map events $\rightharpoonup$ parameters.

Semantics is relative to an environment σ:

$$\begin{array}{lll}
(h,i) \models^{\sigma} e(v) & \text{iff} & e \in dom(x_i) \text{ and } x_i(e) = \sigma(v) \\
 & \vdots & \\
(h,i) \models^{\sigma} \forall x : P_j.\psi & \text{iff} & \text{for all } p \in P_j.(h,i) \models^{((x \mapsto p)/\sigma)} \psi \\
(h,i) \models^{\sigma} \exists x : P_j.\psi & \text{iff} & \text{there exists } p \in P_j.(h,i) \models^{((x \mapsto p)/\sigma)} \psi
\end{array}$$

Verifying Quantified Policies

Given history h and quantified policy ψ, does $h \models \psi$?

We can generalise boolean array algorithm by:

- Eliminating quantifiers by (careful) instantiation of variables
- Binding variables to parameters via a constraints language

Constraints:

$$c ::= \bot \mid x = p \mid c \wedge c \mid c \vee c \mid \neg c \qquad (x \in Var, p \in Par)$$

We map (h, k, ψ) into a constraint $[\![\psi]\!]_h^k$; e.g.,

$$[\![e(v)]\!]_h^k = \begin{cases} x = p & \text{if} \quad v = x \text{ and } h_k(e) = p; \\ \top & \text{if} \quad v = p \text{ and } h_k(e) = p; \\ \bot & \text{if} \quad \text{otherwise} \end{cases}$$

$[\![Qx : P.\psi]\!]_h^k$ is obtained by a conjuction/disjunction of constraints over all possible instantiations of x: there are only finitely many!

Verifying Quantified Policies

Given history h and quantified policy ψ, does $h \models \psi$?

We can generalise boolean array algorithm by:

- Eliminating quantifiers by (careful) instantiation of variables
- Binding variables to parameters via a constraints language

Constraints:

$$c ::= \bot \mid x = p \mid c \wedge c \mid c \vee c \mid \neg c \qquad (x \in Var, p \in Par)$$

We map (h, k, ψ) into a constraint $[\![\psi]\!]_h^k$; e.g.,

$$[\![e(v)]\!]_h^k = \begin{cases} x = p & \text{if} \quad v = x \text{ and } h_k(e) = p; \\ \top & \text{if} \quad v = p \text{ and } h_k(e) = p; \\ \bot & \text{if} \quad \text{otherwise} \end{cases}$$

$[\![Qx : P.\psi]\!]_h^k$ is obtained by a conjuction/disjunction of constraints over all possible instantiations of x: there are only finitely many!

Verifying Quantified Policies

Given history h and quantified policy ψ, does $h \models \psi$?

We can generalise boolean array algorithm by:

- Eliminating quantifiers by (careful) instantiation of variables
- Binding variables to parameters via a constraints language

Constraints:

$$c ::= \bot \mid x = p \mid c \wedge c \mid c \vee c \mid \neg c \qquad (x \in Var, p \in Par)$$

We map (h, k, ψ) into a constraint $[\![\psi]\!]_h^k$; e.g.,

$$[\![e(v)]\!]_h^k = \begin{cases} x = p & \text{if} \quad v = x \text{ and } h_k(e) = p; \\ \top & \text{if} \quad v = p \text{ and } h_k(e) = p; \\ \bot & \text{if} \quad \text{otherwise} \end{cases}$$

$[\![Qx : P.\psi]\!]_h^k$ is obtained by a conjuction/disjunction of constraints over all possible instantiations of x: there are only finitely many!

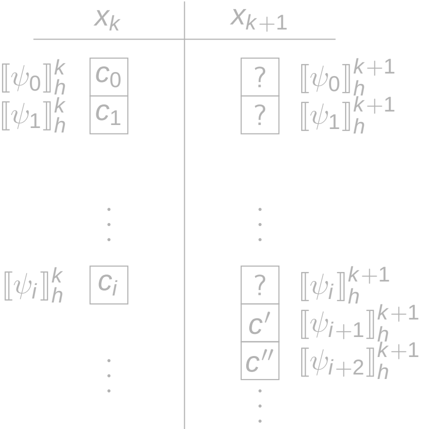
Constraint-Array Algorithm

Maintain

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

Invariant

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



Algorithm – case S

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

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

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

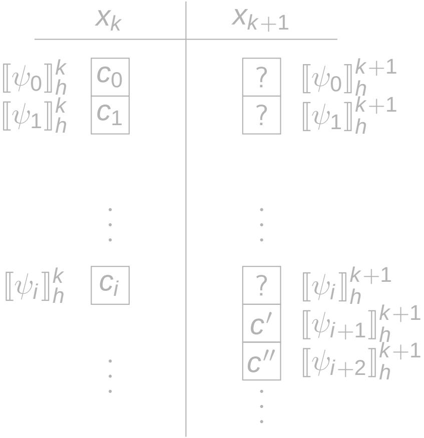
Constraint-Array Algorithm

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 $\psi_i = \psi_{i+1} \; \mathsf{S} \; \psi_{i+2}$
then we can define

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

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

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

	x_k	x_{k+1}	
$[\![\psi_0]\!]^k_h$	c_0	?	$[\![\psi_0]\!]^{k+1}_h$
$[\![\psi_1]\!]^k_h$	c_1	?	$[\![\psi_1]\!]^{k+1}_h$
	$\vdots$	$\vdots$	
$[\![\psi_i]\!]^k_h$	c_i	?	$[\![\psi_i]\!]^{k+1}_h$
		c'	$[\![\psi_{i+1}]\!]^{k+1}_h$
		c''	$[\![\psi_{i+2}]\!]^{k+1}_h$
	$\vdots$	$\vdots$	

Algorithm – case S

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

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

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

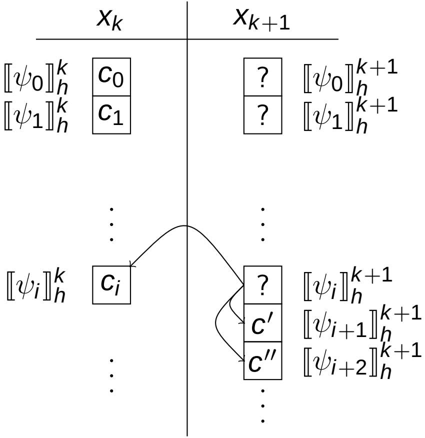
Constraint-Array Algorithm

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 $\psi_i = \psi_{i+1} \; \mathsf{S} \; \psi_{i+2}$
then we can define

$$C_{k+1}[i] \;=\; C_{k+1}[i+2] \vee (C_k[i] \wedge C_{k+1}[i+1])$$

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

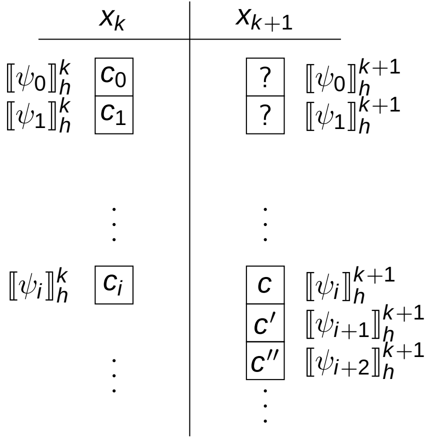
Constraint-Array Algorithm

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 $\psi_i = \psi_{i+1} \; \mathsf{S} \; \psi_{i+2}$
then we can define

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

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

Complexity results

Theorem

Model checking of quantified policies is decidable.

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

- *Proof: reduction from quantified boolean logic)*

Theorem

DMC.**init**()	$O(\|\psi\|)$
DMC.**new**()	$O(\|\psi\| \cdot (\|P_h\| + 1)^n)$
DMC.**update**(e, p, i)	$O((K - i + 1) \cdot \|\psi\| \cdot (\|P_h\| + 2)^n)$
DMC.**check**()	$O(1)$

(n number of quantifiers)

Complexity results

Theorem

Model checking of quantified policies is decidable.

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

- *Proof: reduction from quantified boolean logic)*

Theorem

$DMC.\mathbf{init}()$	$O(\lvert\psi\rvert)$
$DMC.\mathbf{new}()$	$O(\lvert\psi\rvert \cdot (\lvert P_h\rvert + 1)^n)$
$DMC.\mathbf{update}(e, p, i)$	$O((K - i + 1) \cdot \lvert\psi\rvert \cdot (\lvert P_h\rvert + 2)^n)$
$DMC.\mathbf{check}()$	$O(1)$

(n number of quantifiers)

Complexity results

Theorem

Model checking of quantified policies is decidable.

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

- *Proof: reduction from quantified boolean logic)*

Theorem

$DMC.\textbf{init}()$	$O(\lvert\psi\rvert)$
$DMC.\textbf{new}()$	$O(\lvert\psi\rvert \cdot (\lvert P_h\rvert + 1)^n)$
$DMC.\textbf{update}(e, p, i)$	$O((K - i + 1) \cdot \lvert\psi\rvert \cdot (\lvert P_h\rvert + 2)^n)$
$DMC.\textbf{check}()$	$O(1)$

(n number of quantifiers)

References

Add references to other principal's observations.

$$\pi \quad ::= \quad p : \psi \mid \pi_0 \wedge \pi_1 \mid \neg \pi \qquad p \in Prin$$

Policy $p : \psi$ expresses that p's observations satisfy ψ.

eBay revisited: Requirement by p – *"seller has never provided negative feedback in auctions where I made payment, and has never cheated me or any of my friends."*

$$\pi_p^{\text{bid}} \equiv \quad p : \mathsf{G}^{-1}(\texttt{negative} \rightarrow \texttt{ignore}) \quad \wedge$$
$$\bigwedge_{q \in \{p, p_1, \ldots, p_n\}} q : \neg \mathsf{F}^{-1}(\texttt{time-out})$$

References

Add references to other principal's observations.

$$\pi \quad ::= \quad p : \psi \mid \pi_0 \wedge \pi_1 \mid \neg \pi \qquad p \in \mathit{Prin}$$

Policy $p : \psi$ expresses that p's observations satisfy ψ.

eBay revisited: Requirement by p – *"seller has never provided negative feedback in auctions where I made payment, and has never cheated me or any of my friends."*

$$\pi_p^{\text{bid}} \equiv \quad p : \mathsf{G}^{-1}(\texttt{negative} \rightarrow \texttt{ignore}) \quad \wedge \\ \bigwedge_{q \in \{p, p_1, \ldots, p_n\}} q : \neg \mathsf{F}^{-1}(\texttt{time-out})$$

Quantitative Properties

Add event counting to the language.

$$\psi \quad ::= \quad \ldots \mid \mathcal{R}(\overline{\#}\psi_1, \overline{\#}\psi_2, \ldots, \overline{\#}\psi_k)$$

$\overline{\#}\psi$ denotes the number of times ψ has been true in the current sessions, $\mathcal{R}$ is a computable predicate on integers.

P2P File-sharing: We express a policy used by server p for granting download, *"the number of uploads should be at least a third of the number of downloads."*

$$\pi_p^{\text{client-dl}} \equiv p : (\overline{\#}\mathtt{dl} \leq 3 \cdot \overline{\#}\mathtt{ul})$$

"Frequency" policy: We express that *"statistically, event $ev \in E$ occurs with frequency at least 75%."*

$$\pi_p^{\text{probab}} \equiv p : \frac{\overline{\#}\mathtt{ev}}{\overline{\#}\mathtt{ev} + \overline{\#}(\neg\mathtt{ev} \wedge \neg\Diamond\mathtt{ev}) + 1} \geq 3/4$$

Quantitative Properties

Add event counting to the language.

$$\psi \quad ::= \quad \ldots \mid \mathcal{R}(\overline{\#}\psi_1, \overline{\#}\psi_2, \ldots, \overline{\#}\psi_k)$$

$\overline{\#}\psi$ denotes the number of times ψ has been true in the current sessions, $\mathcal{R}$ is a computable predicate on integers.

P2P File-sharing: We express a policy used by server p for granting download, *"the number of uploads should be at least a third of the number of downloads."*

$$\pi_p^{\text{client-dl}} \equiv p : (\overline{\#}\mathtt{dl} \leq 3 \cdot \overline{\#}\mathtt{ul})$$

"Frequency" policy: We express that *"statistically, event $ev \in E$ occurs with frequency at least 75%."*

$$\pi_p^{\text{probab}} \equiv p : \frac{\overline{\#}\mathtt{ev}}{\overline{\#}\mathtt{ev} + \overline{\#}(\neg\mathtt{ev} \wedge \neg\Diamond\mathtt{ev}) + 1} \geq 3/4$$

Quantitative Properties

Add event counting to the language.

$$\psi \quad ::= \quad \ldots \mid \mathcal{R}(\overline{\#}\psi_1, \overline{\#}\psi_2, \ldots, \overline{\#}\psi_k)$$

$\overline{\#}\psi$ denotes the number of times ψ has been true in the current sessions, $\mathcal{R}$ is a computable predicate on integers.

P2P File-sharing: We express a policy used by server p for granting download, *"the number of uploads should be at least a third of the number of downloads."*

$$\pi_p^{\text{client-dl}} \equiv p : (\overline{\#}\texttt{dl} \leq 3 \cdot \overline{\#}\texttt{ul})$$

"Frequency" policy: We express that *"statistically, event $ev \in E$ occurs with frequency at least 75%."*

$$\pi_p^{\text{probab}} \equiv p : \frac{\overline{\#}\text{ev}}{\overline{\#}\text{ev} + \overline{\#}(\neg\text{ev} \wedge \neg\Diamond\text{ev}) + 1} \geq 3/4$$

Summary

- A framework for "reputation systems" and a notion of "security" for such systems.
  - applications to history-based access control.
- Basic Policies can be specified declaratively and verified efficiently.
- Quantified policies are expressive, and quantified model checking is decidable (though hard with many quantifiers).
- Future Work?
  - Tighten bounds on quantified algorithm