

# Distributed Approximation of Fixed Points in Trust Structures (Extended Abstract)

Karl Krukow<sup>1</sup>   Andrew Twigg<sup>2</sup>

<sup>1</sup>BRICS, Department of Computer Science  
University of Aarhus, Denmark

<sup>2</sup>Computer Lab  
University of Cambridge, UK

International Conference on Distributed Computing Systems, 2005

# Setting

- The theme of this work is *trust management* (TM) systems (coined in the PolicyMaker system of Blaze et al. (1996)).
  - ▶ Supports security decision-making in open, large-scale distributed applications.
- Key Concept:
  - ▶ Security Policy.
  - ▶ A corresponding notion of compliance with security policy.
- Foundations of TM has been the subject of sophisticated theoretical work.
  - ▶ (e.g., Mitchell; Li; Feigenbaum; Weeks; Shmatikov & Talcott; Carbone, Nielsen & Sassone; others ...)

# Motivation

- Our work completes the trust-structure framework: a general semantic model for trust.
  - ▶ M. Carbone, M. Nielsen and V. Sassone. *A Formal Model for Trust in Dynamic Networks* (2003).
- In line with the original ideas of TM.
  - ▶ Based on decentralized “trust-policies”.
  - ▶ Flexible.
  - ▶ Separates mechanism from policy (infact, **there is no mechanism at all!**)
- This last point is exactly the topic of this work.
  - ▶ Augment the framework with **operational mechanisms** for policy-based decision-making.

# Outline

## 1 Trust Structures

- A Soft Introduction
- The Formal Model
- The Operational Problem

## 2 Our Contribution

- A Distributed Algorithm for Computing Trust Values
- Approximation protocols
- Dynamic Policy Updates

# Outline

## 1 Trust Structures

- A Soft Introduction
- The Formal Model
- The Operational Problem

## 2 Our Contribution

- A Distributed Algorithm for Computing Trust Values
- Approximation protocols
- Dynamic Policy Updates

# Trust Structures: An Introduction

- Provides a generic mathematical framework, formalizing and solving the following problem.
  - ▶ Given a set  $\mathcal{P}$  of principal identities, each specifying a local trust-policy, define a **unique global trust-state** compatible with those policies.
- A *global trust-state*?
  - ▶ formally, a *global trust state* is a function  $\mathbf{gts} : \mathcal{P} \rightarrow \mathcal{P} \rightarrow X = \mathbf{GTS}$  for some set  $X$  of “trust degrees” (called trust values).
  - ▶ for each  $p, q \in \mathcal{P}$  answer: “to what degree does  $p$  trust  $q$ ?” as  $\mathbf{gts}(p)(q)$ .
- A *generic* model?
  - ▶ Different applications have different requirements for trust-information.
  - ▶ Obtained by choosing set  $X$  of trust degrees, and by choosing appropriate trust policies.

# Trust Structures: An Introduction

- Provides a generic mathematical framework, formalizing and solving the following problem.
  - ▶ Given a set  $\mathcal{P}$  of principal identities, each specifying a local trust-policy, define a **unique global trust-state** compatible with those policies.
- A *global trust-state*?
  - ▶ formally, a *global trust state* is a function  $\mathbf{gts} : \mathcal{P} \rightarrow \mathcal{P} \rightarrow X = \mathbf{GTS}$  for some set  $X$  of “trust degrees” (called trust values).
  - ▶ for each  $p, q \in \mathcal{P}$  answer: “to what degree does  $p$  trust  $q$ ?” as  $\mathbf{gts}(p)(q)$ .
- **Trust policies**?
  - ▶ Specifies how a principal defines its trust in others.
  - ▶  $\pi_p$ : “my trust in:- Alice is **high**, -Bob is **low** - for anyone else, it is the maximum of what Alices thinks and **unknown**.”
  - ▶ Formally:  $\left[ \mathbf{gts} \right] \xrightarrow{\pi_p} \left[ \mathbf{Its} \right]$   
i.e.  $\pi_p : (\mathcal{P} \rightarrow \mathcal{P} \rightarrow X) \rightarrow \mathcal{P} \rightarrow X$

# Trust Structures: The Formal Model (1/3)

- Refining the main objective of the model.
  - ▶ Fix a set  $\mathcal{P}$  of principal identities.
  - ▶ Each principal  $p \in \mathcal{P}$  specifying a local trust-policy  $\pi_p$ .  
Let  $\Pi = (\pi_p : \mathbf{GTS} \rightarrow \mathbf{LTS} \mid p \in \mathcal{P})$  be the (global) collection of all these policies.
  - ▶ Goal is to define a suitable **unique** *global trust-state*, denoted  $\overline{\mathbf{gts}}_{\Pi}$ , respecting the locality of control for all policies  $\Pi$ .
- But... policies may have cyclic *references*.
  - ▶ Example:  $\pi_p(\mathbf{gts}) = \mathbf{gts}(q)$ ,  $\pi_q(\mathbf{gts}) = \mathbf{gts}(p)$ .
  - ▶ In this example, the policies contain *no information*: this is distinct from explicitly specifying *untrusted*.
- This example suggests that the set  $X$  of trust-values can be ordered in two fundamentally distinct ways: with respect to
  - ▶ trust / privilege ( $\preceq$ ), e.g. low  $\preceq$  high
  - ▶ information content ( $\sqsubseteq$ ), e.g. unknown  $\sqsubseteq$  high

# Trust Structures: The Formal Model (1/3)

- Refining the main objective of the model.
  - ▶ Fix a set  $\mathcal{P}$  of principal identities.
  - ▶ Each principal  $p \in \mathcal{P}$  specifying a local trust-policy  $\pi_p$ .  
Let  $\Pi = (\pi_p : \mathbf{GTS} \rightarrow \mathbf{LTS} \mid p \in \mathcal{P})$  be the (global) collection of all these policies.
  - ▶ Goal is to define a suitable **unique** *global trust-state*, denoted  $\overline{\mathbf{gts}}_{\Pi}$ , respecting the locality of control for all policies  $\Pi$ .
- But... policies may have cyclic *references*.
  - ▶ Example:  $\pi_p(\mathbf{gts}) = \mathbf{gts}(q)$ ,  $\pi_q(\mathbf{gts}) = \mathbf{gts}(p)$ .
  - ▶ In this example, the policies contain *no information*: this is distinct from explicitly specifying *untrusted*.
- This example suggests that the set  $X$  of trust-values can be ordered in two fundamentally distinct ways: with respect to
  - ▶ trust / privilege ( $\preceq$ ), e.g. low  $\preceq$  high
  - ▶ information content ( $\sqsubseteq$ ), e.g. unknown  $\sqsubseteq$  high

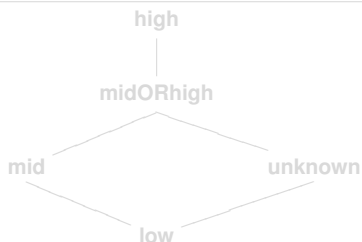
# Trust Structures: The Formal Model (2/3)

## Definition [Trust Structure]

A *trust structure* is a triple  $T = (X, \preceq, \sqsubseteq)$ , consisting of a set  $X$  and two orderings on  $X$ , called the *trust ordering* ( $\preceq$ ) and the *information ordering* ( $\sqsubseteq$ ). The trust ordering preorders  $X$  and the information order makes  $(X, \sqsubseteq)$  a complete partial order with a least element  $\perp_{\sqsubseteq}$ .

- Example  $X = \{\text{low}, \text{mid}, \text{high}, \text{unknown}, \text{midORhigh}\}$ .

Trust order,  $\preceq$



Information Order,  $\sqsubseteq$



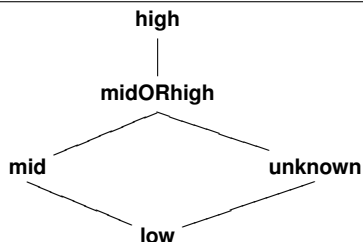
# Trust Structures: The Formal Model (2/3)

## Definition [Trust Structure]

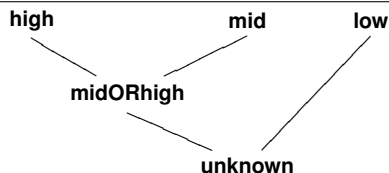
A *trust structure* is a triple  $T = (X, \preceq, \sqsubseteq)$ , consisting of a set  $X$  and two orderings on  $X$ , called the *trust ordering* ( $\preceq$ ) and the *information ordering* ( $\sqsubseteq$ ). The trust ordering preorders  $X$  and the information order makes  $(X, \sqsubseteq)$  a complete partial order with a least element  $\perp_{\sqsubseteq}$ .

- Example  $X = \{\mathbf{low}, \mathbf{mid}, \mathbf{high}, \mathbf{unknown}, \mathbf{midORhigh}\}$ .

Trust order,  $\preceq$



Information Order,  $\sqsubseteq$



## Trust Structures: The Formal Model (3/3)

- Let  $T = (X, \preceq, \sqsubseteq)$  be a trust structure, and  $\Pi = (\pi_p \mid p \in \mathcal{P})$  be a collection of trust policies.
- A *fixed point* of  $\Pi$  is a global trust-state **gts** so that for all  $p \in \mathcal{P}$  we have  $\pi_p(\mathbf{gts})(q) = \mathbf{gts}(p)(q)$ .
  - ▶ Any fixed point of  $\Pi$  is *consistent with all policies*.
- Assuming that all policies are *continuous with respect to*  $\sqsubseteq$ .

### Definition $\overline{\mathbf{gts}}_\Pi$

Define the global trust-state of  $\Pi$  in  $T$  as the  $\sqsubseteq$ -least fixed-point of  $\Pi$ .

$$\overline{\mathbf{gts}}_\Pi \stackrel{(def)}{=} \text{lfp}_{\sqsubseteq} \Pi$$

## Trust Structures: The Formal Model (3/3)

- Let  $T = (X, \preceq, \sqsubseteq)$  be a trust structure, and  $\Pi = (\pi_p \mid p \in \mathcal{P})$  be a collection of trust policies.
- A *fixed point* of  $\Pi$  is a global trust-state **gts** so that for all  $p \in \mathcal{P}$  we have  $\pi_p(\mathbf{gts})(q) = \mathbf{gts}(p)(q)$ .
  - ▶ Any fixed point of  $\Pi$  is *consistent with all policies*.
- Assuming that all policies are *continuous with respect to*  $\sqsubseteq$ .

### Definition $[\overline{\mathbf{gts}}_\Pi]$

Define the global trust-state of  $\Pi$  in  $T$  as the  $\sqsubseteq$ -least fixed-point of  $\Pi$ .

$$\overline{\mathbf{gts}}_\Pi \stackrel{(def)}{=} \text{lfp}_{\sqsubseteq} \Pi$$

# The Operational Problem (1/2)

- Suppose principal  $p$  has to make a *trust-based* access-control decision about another principal  $q$ .
- A common **security policy** for  $p$  would be
  - ▶ Allow access to  $q$  if my trust in  $q$  is trust-wise above threshold  $t \in X$ . (e.g.,  $t = \mathbf{high}$ )
- Generally, a *trust-based* security policy is a function of **the trust in the requestor** ( $q$ ).
- So one *mechanism* for deciding a request could be
  - ▶ compute  $x := \overline{\mathbf{gts}}_{\square}(p)(q)$
  - ▶ compare this value with threshold,  $x \preceq \mathbf{high}$ .

# The Operational Problem (1/2)

- Suppose principal  $p$  has to make a *trust-based* access-control decision about another principal  $q$ .
- A common **security policy** for  $p$  would be
  - ▶ Allow access to  $q$  if my trust in  $q$  is trust-wise above threshold  $t \in X$ . (e.g.,  $t = \mathbf{high}$ )
- Generally, a *trust-based* security policy is a function of **the trust in the requestor** ( $q$ ).
- So one *mechanism* for deciding a request could be
  - ▶ compute  $x := \overline{\mathbf{gts}}_{\Pi}(p)(q)$
  - ▶ compare this value with threshold,  $x \preceq \mathbf{high}$ .

# The Operational Problem (1/2)

- Suppose principal  $p$  has to make a *trust-based* access-control decision about another principal  $q$ .
- A common **security policy** for  $p$  would be
  - ▶ Allow access to  $q$  if my trust in  $q$  is trust-wise above threshold  $t \in X$ . (e.g.,  $t = \mathbf{high}$ )
- Generally, a *trust-based* security policy is a function of **the trust in the requestor** ( $q$ ).
- So one *mechanism* for deciding a request could be
  - ▶ compute  $x := \overline{\mathbf{gts}}_{\Pi}(p)(q)$
  - ▶ compare this value with threshold,  $x \preceq \mathbf{high}$ .

# The Operational Problem (1/2)

- Suppose principal  $p$  has to make a *trust-based* access-control decision about another principal  $q$ .
- A common **security policy** for  $p$  would be
  - ▶ Allow access to  $q$  if my trust in  $q$  is trust-wise above threshold  $t \in X$ . (e.g.,  $t = \mathbf{high}$ )
- Generally, a *trust-based* security policy is a function of **the trust in the requestor** ( $q$ ).
- So one *mechanism* for deciding a request could be
  - ▶ compute  $x := \overline{\mathbf{gts}}_{\Pi}(p)(q)$
  - ▶ compare this value with threshold,  $x \preceq \mathbf{high}$ .

## The Operational Problem (2/2)

- Given  $\Pi$ ,  $\overline{\mathbf{gts}}_{\Pi}$  is defined as the least fixed point of  $\Pi$ .
- A **denotational** model:
  - ▶  $\overline{\mathbf{gts}}_{\Pi}$  is a well defined, unique mathematical object.
- But there is “no recipe for getting there”?
  - ▶ One might say there is no mechanism for evaluating policies.
- How do we actually *compute* the trust values, when
  - ▶  $\Pi$  is distributed.
  - ▶  $|\mathcal{P}|$  is large.
  - ▶ no centralized authority.
  - ▶ policy updates . . .
- The standard technique for fixed point computation:  
 $\perp_{\square} \subseteq \Pi(\perp_{\square}) \subseteq \Pi^2(\perp_{\square}) \cdots \subseteq \Pi^{|\mathcal{P}|^2 \cdot h}(\perp_{\square})?$ 
  - ▶ Inadequate!

## The Operational Problem (2/2)

- Given  $\Pi$ ,  $\overline{\mathbf{gts}}_{\Pi}$  is defined as the least fixed point of  $\Pi$ .
- A **denotational** model:
  - ▶  $\overline{\mathbf{gts}}_{\Pi}$  is a well defined, unique mathematical object.
- But there is “no recipe for getting there”?
  - ▶ One might say there is no mechanism for evaluating policies.
- How do we actually *compute* the trust values, when
  - ▶  $\Pi$  is distributed.
  - ▶  $|\mathcal{P}|$  is large.
  - ▶ no centralized authority.
  - ▶ policy updates . . .
- The standard technique for fixed point computation:  
 $\perp_{\square} \subseteq \Pi(\perp_{\square}) \subseteq \Pi^2(\perp_{\square}) \cdots \subseteq \Pi^{|\mathcal{P}|^2 \cdot h}(\perp_{\square})$ ?
  - ▶ Inadequate!

# Outline

- 1 Trust Structures
  - A Soft Introduction
  - The Formal Model
  - The Operational Problem

- 2 Our Contribution
  - A Distributed Algorithm for Computing Trust Values
  - Approximation protocols
  - Dynamic Policy Updates

# A Distributed Asynchronous Algorithm

- Scenario.

- ▶ Principal  $p$  needs to compute its trust in  $q$ , i.e., the value  $\overline{\mathbf{gts}}_{\Pi}(p)(q)$ .
- ▶ Principals are network nodes with computational and communication capacity.
  - ★ Asynchronous, but reliable network.

- Basic observation.

- ▶ Compute the *local* value  $\overline{\mathbf{gts}}_{\Pi}(p)(q)$  directly, rather than computing the global state  $\overline{\mathbf{gts}}_{\Pi}$  and then looking up “entry”  $(p, q)$ .
- ▶ Excludes principals that are not relevant for the specific computation.

- Algorithm: two-step computation.

- ▶ Dependency analysis distributedly computes a sub-graph  $G_{(p,q)}$  of the dependency graph  $G$  for the policies  $\Pi$ .
- ▶ Asynchronous fixed-point algorithm in dependency graph.

# A Least-Fixed-Point Algorithm (1/2)

- Very simple algorithm; instance of a framework for asynchronous fixed-point algorithms (Bertsekas and Tsitsiklis, 1989).
- Tailored for local **least** fixed-points ( $\overline{\mathbf{gts}}_{\Pi}(p)(q)$ ).
- Essentially a distributed, asynchronous version of the synchronous iteration:

$$\perp_{\square} \sqsubseteq \Pi(\perp_{\square}) \sqsubseteq \dots \sqsubseteq \Pi^i(\perp_{\square})$$

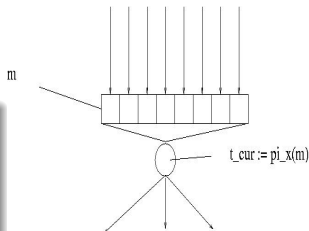
- Proved correct using the **Asynchronous Convergence Theorem** of Bertsekas; the following invariant is essentially maintained:
  - ▶ Node  $p$ , is able to compute a sequence of values  $\perp_{\square} \sqsubseteq t_0 \sqsubseteq \dots \sqsubseteq t_n = \overline{\mathbf{gts}}_{\Pi}(p)(q)$  converging towards its local fixed point value.
  - ▶ Similar convergence holds at other nodes.

# A Least-Fixed-Point Algorithm (2/2)

Node  $x \in G_{(p,q)}$  is always *asleep* or *awake*.

## Algorithm [(essential) State]

- Outgoing  $x^+$  and ingoing  $x^-$  edges of  $x$  in  $G_{(p,q)}$  (dependencies).
- Array  $x.m$  of type  $X$  (storing trust values), indexed by  $x^+$ .
- Variables  $x.t_{old}$  and  $x.t_{cur}$  of type  $X$ .



## Algorithm [Transitions]

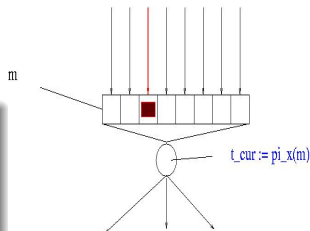
```
input    recv(x,q,t)
         effect x.m[q] := t; wake := true;
output   send(x,q,t)
         precondition (not wake) and (t_cur = t) and (send[q] = true)
         effect send[q] := false
internal eval(id)
         precondition wake
         effect wake := false;
           x.t_old := x.t_cur;
           x.t_cur := pi_x[x.m];
           if x.t_old != x.t_cur
           then
             for q : Principal in x^- do send[q] := true od
           fi
```

# A Least-Fixed-Point Algorithm (2/2)

Node  $x \in G_{(p,q)}$  is always *asleep* or *awake*.

## Algorithm [(essential) State]

- Outgoing  $x^+$  and ingoing  $x^-$  edges of  $x$  in  $G_{(p,q)}$  (dependencies).
- Array  $x.m$  of type  $X$  (storing trust values), indexed by  $x^+$ .
- Variables  $x.t_{old}$  and  $x.t_{cur}$  of type  $X$ .



## Algorithm [Transitions]

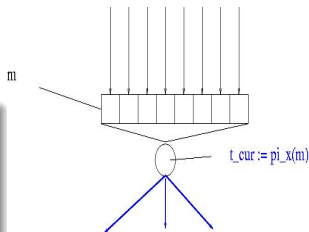
```
input   recv(x,q,t)
        effect x.m[q] := t; wake := true;
output  send(x,q,t)
        precondition (not wake) and (t_cur = t) and (send[q] = true)
        effect send[q] := false
internal eval(id)
        precondition wake
        effect wake := false;
          x.t_old := x.t_cur;
          x.t_cur := pi_x[x.m];
          if x.t_old != x.t_cur
            then
              for q : Principal in x^- do send[q] := true od
            fi
```

# A Least-Fixed-Point Algorithm (2/2)

Node  $x \in G_{(p,q)}$  is always *asleep* or *awake*.

## Algorithm [(essential) State]

- Outgoing  $x^+$  and ingoing  $x^-$  edges of  $x$  in  $G_{(p,q)}$  (dependencies).
- Array  $x.m$  of type  $X$  (storing trust values), indexed by  $x^+$ .
- Variables  $x.t_{old}$  and  $x.t_{cur}$  of type  $X$ .



## Algorithm [Transitions]

```
input    recv(x,q,t)
         effect x.m[q] := t; wake := true;
output   send(x,q,t)
         precondition (not wake) and (t_cur = t) and (send[q] = true)
         effect send[q] := false
internal eval(id)
         precondition wake
         effect wake := false;
           x.t_old := x.t_cur;
           x.t_cur := pi_x[x.m];
           if x.t_old != x.t_cur
           then
             for q : Principal in x^- do send[q] := true od
           fi
```

# Approximation Protocols

- Trust-based security policies
  - ▶ Make security decisions based on trust-values.
  - ▶ A policy is *monotonic* (with respect to  $\preceq$ ), if whenever a value  $t \in X$  grants access, so does any value  $t'$  with  $t \preceq t'$ .
    - ★ e.g., threshold policies.
- Often the exact trust-value is not important!
  - ▶ Instead knowing a property of this value is sufficient to make a decision (e.g., value is above **mid**).
- The idea in our *approximation* protocols is to compute a **safe** approximation of the “real” trust value.
  - ▶ i.e., an approximant  $\bar{p}$  so that  $\bar{p}(p)(q) \preceq \overline{\text{gts}}_{\Pi}(p)(q)$ .

# Approximation Protocols

- Trust-based security policies
  - ▶ Make security decisions based on trust-values.
  - ▶ A policy is *monotonic* (with respect to  $\preceq$ ), if whenever a value  $t \in X$  grants access, so does any value  $t'$  with  $t \preceq t'$ .
    - ★ e.g., threshold policies.
- Often the exact trust-value is not important!
  - ▶ Instead knowing a property of this value is sufficient to make a decision (e.g., value is above **mid**).
- The idea in our *approximation* protocols is to compute a **safe** approximation of the “real” trust value.
  - ▶ i.e., an approximant  $\bar{p}$  so that  $\bar{p}(p)(q) \preceq \overline{\mathbf{gts}}_{\Pi}(p)(q)$ .

# A Proof-Carrying Approximation Protocol

- The idea in the “proof-carrying” approximation protocol is
  - ▶ Avoid an entire fixed-point computation, by having the requestor supply information (“proof”) helping the provider make its security decision.
- Structure of “proof-carrying” authorization: ‘Prover’  $p$  want to access a resource controlled by ‘verifier’  $v$ .
  - ▶ Prover  $p$  (somehow) knows a “proof”/reason  $\bar{p}$  that access should be granted – this is sent to  $v$ .
  - ▶ Verifier, getting help from its dependencies, verifies that  $\bar{p}$  is a “correct” proof.
  - ▶ Access is granted.
- Assumptions:
  - ▶ Security policies are monotonic wrt  $\preceq$ .
  - ▶ Trust policies  $\pi_p$  are monotonic with respect to  $\preceq$ .
  - ▶ (and a few inessential technical assumptions about the trust-structure  $T = (X, \preceq, \sqsubseteq)$ ).

# A Proof-Carrying Approximation Protocol

- The idea in the “proof-carrying” approximation protocol is
  - ▶ Avoid an entire fixed-point computation, by having the requestor supply information (“proof”) helping the provider make its security decision.
- Structure of “proof-carrying” authorization: ‘Prover’  $p$  want to access a resource controlled by ‘verifier’  $v$ .
  - ▶ Prover  $p$  (somehow) knows a “proof”/reason  $\bar{p}$  that access should be granted – this is sent to  $v$ .
  - ▶ Verifier, getting help from its dependencies, verifies that  $\bar{p}$  is a “correct” proof.
  - ▶ Access is granted.
- Assumptions:
  - ▶ Security policies are monotonic wrt  $\preceq$ .
  - ▶ Trust policies  $\pi_p$  are monotonic with respect to  $\preceq$ .
  - ▶ (and a few inessential technical assumptions about the trust-structure  $T = (X, \preceq, \sqsubseteq)$ ).

# Basic theorem

Let  $(X, \preceq, \sqsubseteq)$  be a trust structure in which  $\preceq$  is  $\sqsubseteq$ -continuous. Let  $\Pi : X^{\mathcal{P}^2} \rightarrow X^{\mathcal{P}^2}$  be any function that is  $\sqsubseteq$ -continuous and  $\preceq$ -monotonic.

## Theorem (Proof Carrying)

Let  $\bar{p} \in X^{\mathcal{P}^2}$ . If we have  $\bar{p} \preceq \perp_{\sqsubseteq}$  and  $\bar{p} \preceq \Pi(\bar{p})$ , then  $\bar{p} \preceq \text{lfp}_{\sqsubseteq} \Pi$ .

Illustrative example: Trust Structure  $(X, \preceq, \sqsubseteq)$ ;

$X = \{(m, n) \mid m, n \in \bar{\mathbb{N}} = \mathbb{N} \cup \{\infty\}\}$

- $(m, n) \in X$  represents a history of  $m + n$  interactions;  $m$  “good” and  $n$  “bad”.
- $(m, n) \sqsubseteq (m', n')$  iff  $m \leq m'$  and  $n \leq n'$ .
- $(m, n) \preceq (m', n')$  iff  $m \leq m'$  and  $n \geq n'$ .

# Basic theorem

Let  $(X, \preceq, \sqsubseteq)$  be a trust structure in which  $\preceq$  is  $\sqsubseteq$ -continuous. Let  $\Pi : X^{\mathcal{P}^2} \rightarrow X^{\mathcal{P}^2}$  be any function that is  $\sqsubseteq$ -continuous and  $\preceq$ -monotonic.

## Theorem (Proof Carrying)

Let  $\bar{p} \in X^{\mathcal{P}^2}$ . If we have  $\bar{p} \preceq \perp_{\sqsubseteq}$  and  $\bar{p} \preceq \Pi(\bar{p})$ , then  $\bar{p} \preceq \text{lfp}_{\sqsubseteq} \Pi$ .

Illustrative example: Trust Structure  $(X, \preceq, \sqsubseteq)$ ;

$$X = \{(m, n) \mid m, n \in \bar{\mathbb{N}} = \mathbb{N} \cup \{\infty\}\}$$

- $(m, n) \in X$  represents a history of  $m + n$  interactions;  $m$  “good” and  $n$  “bad”.
- $(m, n) \sqsubseteq (m', n')$  iff  $m \leq m'$  and  $n \leq n'$ .
- $(m, n) \preceq (m', n')$  iff  $m \leq m'$  and  $n \geq n'$ .

# The Approximation Protocol (1/2)

## An example

- The  $p$  wants to convince  $v$  that  $(0, N) \preceq \overline{\text{gts}}_{\Pi}(v)(p)$ 
  - ▶ bounding the observed number of “bad” interactions.
- Suppose that  $v$ 's policy depends some large set  $S$  of principals, and is so that if  $a$  and  $b$  in  $S$  have a “high” value for  $p$  then this is sufficient for  $v$ .
- By its previous interactions with  $a$  and  $b$ , it knows that  $a$ 's and  $b$ 's values for  $p$  are above  $(0, N)$ .
- To convince  $v$  that  $v$ 's value for  $p$  is also above  $(0, N)$  it sends the following proof.

$$\bar{p} = [(v, p) \mapsto (0, N), (a, p) \mapsto (0, N), (b, p) \mapsto (0, N), \\ (\_, \_) \mapsto \perp_{\preceq} = (0, \infty)]$$

- ▶ Which we can think of as a claim,  $\bar{p}(x, y) \preceq \overline{\text{gts}}_{\Pi}(x, y)$ .

# The Approximation Protocol (1/2)

## An example

- The  $p$  wants to convince  $v$  that  $(0, N) \preceq \overline{\mathbf{gts}}_{\Pi}(v)(p)$ 
  - ▶ bounding the observed number of “bad” interactions.
- Suppose that  $v$ 's policy depends some large set  $S$  of principals, and is so that if  $a$  and  $b$  in  $S$  have a “high” value for  $p$  then this is sufficient for  $v$ .
- By its previous interactions with  $a$  and  $b$ , it knows that  $a$ 's and  $b$ 's values for  $p$  are above  $(0, N)$ .
- To convince  $v$  that  $v$ 's value for  $p$  is also above  $(0, N)$  it sends the following proof.

$$\bar{p} = [(v, p) \mapsto (0, N), (a, p) \mapsto (0, N), (b, p) \mapsto (0, N), \\ (\_, \_) \mapsto \perp_{\preceq} = (0, \infty)]$$

- ▶ Which we can think of as a claim,  $\bar{p}(x, y) \preceq \overline{\mathbf{gts}}_{\Pi}(x, y)$ .

# The Approximation Protocol (2/2)

## An example

$$\bar{p} = [(v, p) \mapsto (0, N), (a, p) \mapsto (0, N), (b, p) \mapsto (0, N), \\ (\_, \_) \mapsto \perp_{\preceq} = (0, \infty)]$$

- If  $v$  can verify  $\bar{p} \preceq \perp_{\square} = (0, 0)$  and  $\bar{p} \preceq \Pi(\bar{p})$  then  $\bar{p} \preceq \overline{\mathbf{gts}}_{\Pi}$ .
- Checking  $\bar{p}(x, y) \preceq (0, 0)$  is easy.
- To check  $\bar{p} \preceq \Pi(\bar{p})$ :
  - ▶ check  $(0, N) \preceq \pi_v(\bar{p})(v, p)$
  - ▶ ask  $a$  and  $b$  to perform similar check (sending proof  $\bar{p}$ ).
- if all checks succeed,  $v$  knows by theorem,  $(0, N) \preceq \overline{\mathbf{gts}}_{\Pi}(v, p)$ .

# Comments

- However
  - ▶ Prover needs to know that  $v$  relies on  $a$  and  $b$  in this specific manner (information about verifiers policies).
  - ▶ Because  $\bar{p} \preceq \perp_{\square}$ , this can usually only prove “not too much bad behaviour.”
- We have also a snap-shot algorithm which doesn't have these restrictions (the cost is that the algorithm might require more communication and computation).

# Summary

- The trust-structure framework is a **denotational** semantic model for trust-management systems.
  - ▶ We have provided an **operational** counterpart.  
(can be formalized, in terms of I/O Automata, as a correspondance between a denotational and an operational semantics for trust policies ).
- An asynchronous algorithm for computing/approximating trust values.
- Several protocols for trust-value approximation.
- Outlook
  - ▶ Automatic theorem-proving and code-generation using the IOA framework for I/O Automata.
  - ▶ This is a purely theoretical contribution – experimental validation is needed.

# Summary

- The trust-structure framework is a **denotational** semantic model for trust-management systems.
  - ▶ We have provided an **operational** counterpart.  
(can be formalized, in terms of I/O Automata, as a correspondance between a denotational and an operational semantics for trust policies ).
- An asynchronous algorithm for computing/approximating trust values.
- Several protocols for trust-value approximation.
- Outlook
  - ▶ Automatic theorem-proving and code-generation using the IOA framework for I/O Automata.
  - ▶ This is a purely theoretical contribution – experimental validation is needed.

# For Further Reading

- Bertsekas, D. P. and Tsitsiklis, J. N.: 1989,  
*Parallel and Distributed Computation: Numerical Methods*,  
Prentice-Hall International Editions, Prentice-Hall, Inc.
- Blaze, M., Feigenbaum, J., and Lacy, J.: 1996,  
in *Proceedings from the 17th Symposium on Security and Privacy*,  
pp 164–173, IEEE Computer Society Press
- Carbone, M., Nielsen, M., and Sassone, V.: 2003,  
in *Proceedings from Software Engineering and Formal Methods*  
(SEFM'03), IEEE Computer Society Press
- Krukow, K. and Twigg, A.: 2005,  
*Distributed Approximation of Fixed-Points in Trust Structures*,  
Technical Report RS-05-6, BRICS, University of Aarhus,  
Available online: <http://www.brics.dk/RS/05/6>