On Foundations for Dynamic Trust Management

Karl Krukow

PhD Progress Report

Department of Computer Science
University of Aarhus
Denmark

On Foundations for Dynamic Trust Management

Karl Krukow*
BRICS†
Department of Computer Science
University of Aarhus‡

May, 2004

Abstract

We introduce the term *dynamic trust management* as a merger of the traditional concept of trust management with the dynamic aspects of reputation: monitoring of entity behaviour and consequently re-evaluation of trusting relationships; as well as allowing uncertain trust values. We present a preliminary mathematical framework which is based on the notion of *trust structures*. The framework is suitable for expressing trust management systems which allow *uncertain trust values*, and we are able to instantiate it to obtain many existing systems. We give an axiomatisation of trust structures, and a notion of structure-preserving functions on these, resulting in a category. The *interval construction* fits in this setting as a functor into our new category, which is part of a coreflection of our category in a category of complete lattices. We present the dynamic trust model of the SECURE project, and show that its trust structure is an instance of our abstract framework. We show how the SECURE notions of *observation* and *outcome* can be modelled uniformly as finite configurations of event structures. This allows us to give a general notion of morphisms of event structures, which can be applied for the transfer of trust- or evidence-information between related SECURE contexts. We consider the operational aspects of our framework, displaying and analysing several distributed least-fixed-point algorithms which can be applied in dynamic trust management systems for computing trust values. In the concluding section we reflect on this work and consider possible future research directions.

Keywords: Trust, reputation, formal and mathematical models, dynamic trust management, computer security, global computing, autonomous decision-making, SECURE project, trust structures, interval construction, poset intervals, distributed fixed point algorithms, Dempster-Shafer theory of evidence, opinion model, subjective logic, event structures.

*email: krukow@brics.dk
†Basic Research in Computer Science (www.brics.dk),
funded by the Danish National Research Foundation.
‡IT-Parken, Åbogade 34,
DK-8200, Århus N., Denmark

Contents

1 Introduction

Traditional trust management systems [1, 2] deal with the so-called compliance-checking problem: given a *request* to perform a certain action, and a set of *credentials*, does the request together with the credentials comply with local security policy? While the concept of trust management has been widely accepted in security communities, and automated trust management systems have been successfully deployed in distributed applications [3, 4], the traditional systems have been found inadequate for *some* applications in at least two respects: they do not allow for any degree of *uncertain information*, and it does not take into account the *dynamics* of trusting relationships [2, 5, 6]. In particular, the Global Computing[1] (GC) vision foresees *vast* numbers of heterogeneous, networked, mobile, autonomous entities[2] interacting in various contexts, seeking to fulfil their respective goals. In GC environments such entities stand to benefit from cooperation, but only if they can assign meaningful privileges to each other.

This progress report is concerned with foundations for *dynamic trust management*. This concept is our own, and is meant to capture a broader notion than the traditional concept of trust management, coined by Blaze, Feigenbaum and Lacy [7] who presented also the first and very general trust management system, PolicyMaker [7, 8]. More specifically, the term *dynamic* trust management can be thought of as a merger of ideas from traditional trust management with the dynamics of reputation-based systems. The essence of reputation-based systems is that the behaviour of principals is recorded so that continuously well-behaving principals are rewarded, whereas misbehaving ones are punished. Many existing reputation schemes are, however, *global* in two distinct understandings. Firstly, each principal is assigned a unique *global* "reputation value". In contrast, trust management systems are governed by *locally* assigned credentials – a principal might have higher privileges with p than with q. Secondly, the reputation value of a principal is given

[1] The concepts of pervasive and ubiquitous computing have also been used in the literature.

[2] Such entities can be users interacting through hardware devices, e.g. PDAs, Internet-connected PCs or mobile phones, but can also be software, e.g. mobile agents acting on behalf of some user or another program. We will use the term *principal* for such an entity.

by some predetermined formula (often called a reputation metric) computed from the "opinions" about its behaviour of *all* other principals. This means that reputation data must be gathered globally, which has the advantage that it maximises the quantity of data, but the disadvantage that it may affect scalability, and as a side-effect may allow for colluding parties of principals to artificially affect their reputation values. The other aspect of dynamic trust management is that of uncertainty or partial specification of trust. In a GC environment it may not always be possible to have *complete information* about a given principal, or its behaviour, and so as a part of a dynamic trust management system we will allow specification of uncertain or imprecise values.

By *foundations* for dynamic trust management, we mean a mathematical framework which identifies and formalises its characteristic properties. Such a framework should ideally abstract away irrelevant details, while still being concrete enough to faithfully instantiate to various existing (and new) systems, and allow for specification and proof of useful security properties. To be also of practical interest, a good framework must also inherently be of an *operational* or *computational* nature. A very nice example of such a framework, which is also the base of our own approach, is that of Stephen Weeks. In his paper on trust management systems [9], Weeks gives a simple formal framework based on the mathematical theory of complete lattices, which he is able to instantiate, obtaining various existing trust management systems, e.g. SPKI/SDSI [10], KeyNote [11, 12] – the successor of PolicyMaker – and various logic-based systems. This work thus enhances the understanding of trust management, facilitates comparison of systems and provides a mental framework for the conception of new systems. Furthermore, by expressing the compliance checking problem essentially as a least fixed point computation and an order-theoretic comparison, Weeks was able to use existing theory of algorithms for least-fixed-point computation to obtain more efficient algorithms for the compliance-checking problem.

As it is the prime source of inspiration for our own approach, we describe here, in some detail, the framework of Weeks. In this framework, a trust management system is expressed as a complete lattice $(D, \leq)$ of possible *authorisations*, a set $\mathcal{P}$ of *principal names*, and a language for specifying so-called *licenses*. The lattice elements $d, e \in D$ express the authorisations relevant for a particular system, e.g. access-rights, and $d \leq e$ then means that e authorises at least as much as d. An *assertion* is a pair $a = \langle p, l \rangle$ consisting of a principal $p \in \mathcal{P}$, the *issuer*, and a monotone function $l : (\mathcal{P} \to D) \to D$, called a *license*. In the simplest case l could be a constant function, say d_0, and then a states that p authorises d_0. In the general case the interpretation of a is: given that all principals authorise as specified in the *authorisation map*, $m : \mathcal{P} \to D$, then p authorises as specified in $l(m)$. This means that a license such as $l(m) = m(A) \vee m(B)$ expresses a policy saying "give the least upper bound in $(D, \leq)$ of what A says and what B says". Weeks showed that a collection of assertions $L = \langle p_i, l_i \rangle_{i \in I}$ gives rise to a monotone function $L_\lambda : (\mathcal{P} \to D) \to \mathcal{P} \to D$, with the property that a coherent authorisation map representing the authorisations of the involved principals is given by the least fixed point, $\mathsf{lfp}\ L_\lambda$.

Nielsen *et al.* argues that while Weeks' idea of having mutually referring licenses resolved by fixed points was good, the Weeks-framework for trust would often be too restrictive in GC environments [5, 6]. One reason for this is that often principals do not have sufficient information to specify precise authorisations for all other principals. In the framework this means that any unknown or only partially known principal is always assigned the bottom authorisation. Authors also proposed a framework which solved this problem by having the set T of "authorisations", here called *trust values*, equipped with *two* orderings, denoted $\preceq$ and $\sqsubseteq$. In this setting, $\preceq$, called the *trust ordering*, corresponds to Weeks' way of ordering by "more

privilege", whereas $\sqsubseteq$, called the *information ordering*, introduces a notion of precision or information. The key idea was that the elements of the set should embody also various degrees of uncertainty, and then $d \sqsubseteq e$ reflects that e is more precise or contains more information than d. In the simplest case, the trust values are just symbolic, e.g. $\mathsf{unknown} \sqsubseteq \mathsf{low} \preceq \mathsf{high}$, but they may also have more internal structure as will be evident from examples in the following sections. It was shown how least fixed points with respect to the *information ordering* leads to a way of distinguishing e.g. an unknown principal from a known and distrusted one.

This report starts by presenting the approach of Nielsen *et al.* [5, 6], which displays a formal framework based on the ideas of Weeks, but which recognises the need for uncertain or partially specified trust information. We investigate further the concept of *trust structures* [5, 6], which are the bi-order structures $(T, \preceq, \sqsubseteq)$, and consider adjoining to these some natural axioms, obtaining a category. The "canonical" construction, known as the *interval construction* [5, 6], provides a constructive method adding to any complete lattice, additional elements containing various degrees of uncertainty. This construction fits beautifully in our setting as the full and faithful left adjoint in a coreflection[3] of our new category in a category of complete lattices, thus giving a categorical account of the construction. We obtain Weeks' framework as special "dogmatic" objects in this category, and our framework also instantiates to many existing trust models, e.g. the trust model of the SECURE[4] project [15, 16, 17]. We display some other instantiations of the framework, and consider the SECURE approach in more depth. The last part of the report is concerned with operational aspects of our approach. We describe and discuss approximation- and exact-distributed least-fixed-point algorithms, enabling the computation of trust values. We conclude with a discussion of some unsatisfactory aspects of our framework, and present some preliminary ideas towards a refinement of the notion of uncertainty in trust structures.

Related Work: The term trust management was first introduced by Blaze *et al.* [7], and following this influential work came many papers and systems related to trust management (e.g. [18, 11, 10, 19]). Grandison and Sloman [2] present a survey of trust in Internet applications anno 2000.

The core of this report deals with the mathematical modelling of trust in computer systems. The closest related work is the framework of Weeks [9], and its extensions for handling uncertainty [5, 6]. The SECURE project [15, 16, 17] aims at providing a framework for decision-making in GC environments, based on the notion of trust. The formal model for trust in SECURE applies event structures, traditionally studied in the theory of semantics of concurrency [20, 14], in a novel way to faithfully model the SECURE notions of *observation* and *outcome*. We present the trust model deployed in SECURE in Section 3 of this report. As will also be discussed in that section, this modelling using event structures is related to the Dempster-Shafer theory of evidence [21, 22], in that the configurations of our event structures correspond to the so-called frames of discernment of the Dempster-Shafer theory. The subjective logic of Jøsang [23] has been suggested and applied as a way of modelling and reasoning about uncertain probabilities, also called "opinions". There is an essentially bijective correspondence between Jøsangs opinion-triples $\omega = (b, u, d)$ and the interval construction applied to the complete lattice of reals $([0, 1], \leq)$, and in this sense, the opinion-model is an instance of our general framework. The intervals of a poset has been studied previously in the context of

[3] The traditional use of the term coreflection is an adjunction in which the left adjoint is an inclusion [13]. Here the concept is to be understood slightly more generally, as has been suggested by Winskel *et al.* [14], meaning simply an adjunction in which the left adjoint is full and faithful.

[4] SECURE: Secure Environment for Collaboration among Ubiquitous Roaming Entities, EU FET-GC IST-2001-32486

computer arithmetic [24].

It is possible, in a way similar to that of Jøsang, to obtain uncertain probabilities on the configurations of event structures. This is related to the notion of probabilistic event structures, studied in the context of domain theory [25] by Varacca *et al.* [26, 27]. The generalised notion of morphism of event structures in Section 3.3 enables a uniform framework for the transfer of trust information between related contexts. To our knowledge there is no previous work enabling, in a general way, the transfer of trust information between related contexts. However, we have still not verified the usefulness of such transfer functions in practical applications.

As we have mentioned already, the notion of reputation-based systems (e.g. [28, 29, 30, 31, 32, 33]) also addresses some of the issues of dynamics we are interested in.

Dimitri Bertsekas [34] has done a substantial body of work on distributed and parallel algorithms, which is the basis for the algorithms of Section 4.

Prerequisites: This report assumes basic knowledge of the theory of partially ordered sets, including the traditional computer science applications of the theory of complete lattices, which can be obtained e.g. in Winskel's excellent book on semantics of programming languages [35]. General lattice theory is covered by Grätzer [36]. Section 2.2 requires some basic knowledge of category theory, which can be covered e.g. by browsing the basic sections of Mac Lane's book [13], the universal arrow to category theory.

Overview: In Section 2 we describe the approach of Nielsen *et al.* [5, 6] generalising Week's lattice-based framework, and give a slightly modified version of the interval construction. We sketch a preliminary definition of a category of trust structures, and present the result that the interval construction is the left adjoint in a coreflection of our category in a category of complete lattices. In Section 3 we describe in detail the trust model of the SECURE project. We discuss also the problem of transferring information between related contexts. In Section 4 we describe distributed algorithms needed to compute trust values, and discuss requirements and properties of such algorithms. Section 5 concludes and discusses possible future research.

2 A mathematical framework

This section explains briefly the approach of Nielsen *et al.* [5, 6] as it grounds the rest of this report. In this framework, trust is something which exists *between* principals. Each application defines a so-called *trust structure*, which consists of a set T of *trust values* together with the two orderings of T, the trust ordering $\preceq$, and the information ordering $\sqsubseteq$. At any time, the *global trust state* of the system can, in principle, be described as a function $\mathsf{gts} : \mathcal{P} \to \mathcal{P} \to T$, with the interpretation that $\mathsf{gts}(p)(q)$ denotes p's trust in q. A crucial requirement is that the information ordering makes $(T, \sqsubseteq)$ a cpo with bottom. Similarly to Weeks' approach, each principal $p \in \mathcal{P}$ defines a so-called *trust policy*, which is an information-continuous function π_p of type $(\mathcal{P} \to \mathcal{P} \to T) \to (\mathcal{P} \to T)$, with the previously described "recursive" interpretation: given that all principals trust as specified by $m : \mathcal{P} \to \mathcal{P} \to T$ then p's trust is specified by $\pi_p(m)$. The collection of all trust policies $\Pi = (\pi_p : p \in \mathcal{P})$ thus "spins a global web of trust", e.g. p's trust in q depends on a's trust in q and p's trust in a, and a's trust in q, in turn, depends on b's and c's which may, in turn, depend on p's, etc. Part of an application specification is to select a subset of such functions to be allowed as policies. This can be done e.g. by giving a *language* for writing policies, such that any policy written in that language

is guaranteed to be continuous. We give an example of such a *trust policy language* in Section 3.

Since $(T, \sqsubseteq)$ is a cpo with bottom, and all policies are information-continuous, there exists a unique information-continuous function $\Pi_\lambda = \langle \pi_p : p \in \mathcal{P} \rangle$, of type $(\mathcal{P} \to \mathcal{P} \to T) \to \mathcal{P} \to \mathcal{P} \to T$ with the property that $\mathtt{Proj}_p \circ \Pi_\lambda = \pi_p$ for all $p \in \mathcal{P}$, where $\mathtt{Proj}_p$ is the p'th projection[5]. This means that for any continuous collection of trust policies, Π, we can define the unique global trust state induced by that collection, as the *least fixed point* of the product function Π_λ, denoted $\mathsf{gts}(\Pi) = \mathtt{lfp}\ \Pi_\lambda$, which satisfies the fixed point equation

$$\mathsf{gts}(\Pi)(p) = \Pi_\lambda(\mathsf{gts}(\Pi))(p) = (\mathtt{Proj}_p \circ \Pi_\lambda)(\mathsf{gts}(\Pi)) = \pi_p(\mathsf{gts}(\Pi))$$

Reading this from the leftmost to the rightmost side, any function $m : \mathcal{P} \to \mathcal{P} \to T$ satisfying this equation, is *consistent* with Π. Consider now two mutually referring functions π_p and π_q, given by $\pi_p = \lambda m.\mathtt{Proj}_q(m)$, and $\pi_q = \lambda m.\mathtt{Proj}_p(m)$. Intuitively there is no information present in these functions, they simply mutually refer to each other. Thus we would like the global trust state induced by these function to take the value $\bot_\sqsubseteq$ for both p and q. This is exactly what is obtained by choosing the *least* fixed point of Π_λ.

A few examples displaying the expressiveness of the framework seem appropriate.

Example 2.1 *(PGP).* The PGP trust model [37], called the "web of trust", deals with the validation of certificates, that is, pairs $(key, name)$, consisting of a cryptographic public key and the identity of a user. The certificates are signed by "certificate authorities" (CAs), which vouch for the authenticity of the $(key, name)$-binding, i.e. that the *name* controls *key* (holds the corresponding private key). In the web-of-trust approach, *any* PGP user can act as a certificate authority, and it is then up to each individual user to decide which other users it will trust as CAs. To model PGP's web of trust, we assume here sets $Names$ and $Keys$, and assume that $Names$ uniquely identifies the users. Take $\mathcal{P} = Names$, i.e. principals are human users. The PGP trust model supports three distinct trust values to be assigned to $(key, name)$-pairs: *valid*, *marginally valid*, and *invalid*. Furthermore, at any time each principal p has three disjoint sets C_p, M_p and U_p of $(key, name)$-pairs. C_p denotes a set of *completely trusted* pairs, M_p a set of *marginally trusted* pairs, and U_p a set of *untrusted* pairs. An *untrusted* pair is simply a $(key, name)$-pair that p has signed itself, and thus believes to be valid, but which is *not* trusted to introduce *new* valid key-pairs. Our framework instantiates to this model by taking trust values to be $T_{PGP} = Keys \to \{\mathsf{invalid}, \mathsf{mvalid}, \mathsf{valid}\}$, with the obvious pointwise trust ordering $\preceq$, and we take also $\sqsubseteq = \preceq$. Each principal p, then has policy π given by: for any $m : \mathcal{P} \to \mathcal{P} \to T_{PGP}$

$$\pi(m) = \lambda q \in \mathcal{P}.\lambda k \in Keys. \begin{cases} \mathsf{valid} & \text{if } (q,k) \in C_p \cup M_p \cup U_p \\ & \text{or } \exists (r, \_) \in C_p : m(r)(q)(k) = \mathsf{valid} \\ & \text{or } \exists (r_1, \_), (r_2, \_) \in M_p : r_1 \neq r_2 \text{ and} \\ & \qquad m(r_1)(q)(k) = m(r_2)(q)(k) = \mathsf{valid} \\ \mathsf{mvalid} & \text{otherwise if } \exists (r, \_) \in M_p : m(r)(q)(k) = \mathsf{valid} \\ \mathsf{invalid} & \text{otherwise} \end{cases}$$

It is not hard to see that this function is information continuous. Furthermore, if all principals p have the same policy π (but specifies sets C_p, M_p, U_p freely), then choosing least fixed points implements the PGP approach of taking a pair to be valid if it is signed by *one completely trusted* CA or *two distinct marginally trusted* CAs.

[5] In the concise language of category theory: the category of cpos and continuous functions has arbitrary products.

Example 2.2 *(Dogmatic structures).* The above is an example of a so-called "dogmatic" trust structure. Recall that Weeks' framework works by defining a complete lattice of authorisations $(D, \leq)$, and considers (a subset of) the monotone functions of type $(\mathcal{P} \to D) \to D$ as "trust policies". Choosing the trivial trust structure $(D, \leq, \leq)$ gives us essentially the same framework[6], by identifying information with trust. We shall return to this (including the name "dogmatic") in Section 2.2.

Example 2.3 *(mn-pairs).* A simple example of a structure with $\sqsubseteq \neq \preceq$ is the following. Consider the set of pairs of natural numbers $\mathbb{N}^2$. The trust values $(m, n) \in \mathbb{N}^2$ should be thought of as the result of having performed $m + n$ experiments, where each outcome is classified as either *good* or *bad*. Then m is the number of good outcomes and n the number of bad outcomes. With this intuition we define the following orderings: for any $(m_0, n_0), (m_1, n_1) \in \mathbb{N}^2$

$$(m_0, n_0) \sqsubseteq (m_1, n_1) \iff (m_0 \leq m_1) \text{ and } (n_0 \leq n_1)$$

and

$$(m_0, n_0) \preceq (m_1, n_1) \iff (m_0 \leq m_1) \text{ and } (n_0 \geq n_1)$$

The orderings are quite intuitive: a value is less trust than another if it has fewer good and more bad experiments, and values $(m_0, n_0) \sqsubseteq (m_1, n_1)$ only if we can start from (m_0, n_0) and then perform some number of additional experiments and end with (m_1, n_1). We complete this structure by adding three elements "bad": $(0, \infty) = \bot_{\preceq}$, "good": $(\infty, 0) = \top_{\preceq}$ and an information top $(\infty, \infty) = \top_{\sqsubseteq}$. The orderings are still given by the equations above, with the obvious extension to ∞. We let T_{MN} denote this extended set, and then both $(T_{MN}, \sqsubseteq)$ and $(T_{MN}, \preceq)$ are complete lattices. Note that this is a trust structure which supports dynamics: updating a trust value after performing an experiment is simply adding one to the appropriate component. The beta reputation system [33] has a reputation metric which derives its reputation values from observations which are (a continuous version of) mn-pairs.

2.1 Intervals: a canonical construction

The *interval construction* [5, 6] takes any complete lattice, and constructs a trust structure with additional elements containing various degrees of uncertainty. The elements in the constructed structure are intervals of elements in the original lattice.

Example 2.4 *(Opinion model).* Consider the complete lattice of real numbers $D = [0, 1]$, ordered the usual way. We think of the elements $d \in D$ as expressing a degree of belief that a certain proposition ϕ is true[7]. Value 1 expresses complete belief that ϕ is true, and 0, complete belief that ϕ is false. Consider the set of sub-intervals $[x, y] \subseteq D$. One can interpret such an interval as an uncertain belief in ϕ: "the value is in $[x, y]$, but I'm uncertain which particular value it is". Narrowing the interval to $[x_0, y_0]$ with $x \leq x_0$ and $y_0 \leq y$, is then a more *confident* "uncertain belief". If one instead increases the boundaries, e.g. $x_0 = x + \epsilon$ and $y_0 = y + \delta$ then one obtains an uncertain belief which has a higher degree of belief in the truth of ϕ. Jøsang's opinion model [23] uses *opinions*, which are triples $\omega_\phi = (b, u, d)$ of non-negative real numbers with the property that $b + u + d = 1$. Here one interprets b as the *belief* that ϕ is true, d as the *disbelief* that ϕ is true, and u as an *uncertainty* about the truth of ϕ. Note the one-to-one correspondence between such ω_ϕ and intervals $[b, b + u]$. Note also that the singleton-intervals are in one-to-one correspondence with the original lattice D. These singleton intervals which contain no uncertainty are called *dogmas* in Jøsang's model.

[6] Up to continuity vs. monotonicity

[7] An alternative interpretation could be a belief that d is the probability that ϕ is true.

We now give a descriptive account of the interval construction which differs from the original exposition [5, 6] in that we obtain a complete lattice with respect to *both* the trust ordering, *and* the information ordering. We prefer this because it leads to a simpler framework (complete lattices instead of cpos and monotone instead of continuous functions as policies), and because it has a nicer mathematical characterisation[8].

Definition 2.1 (Interval). *Let $(D, \leq)$ be a complete lattice. For any $d, e \in D$ with $d \leq e$ the* interval from d to e in D*, denoted by $[d, e]$, is defined by*

$$[d, e] = \{d' \in D \mid d \leq d' \leq e\} \subseteq D$$

We consider also a special interval, denoted $[\top_{\leq}, \bot_{\leq}]$ which is empty unless D is a one-point lattice. Write ID for the set of intervals of $(D, \leq)$, including $[\top_{\leq}, \bot_{\leq}]$.

Definition 2.2 ($\preceq$). *Define an ordering, $\preceq$, on intervals by: for any pair of intervals $[d, e]$, $[d', e']$ we have*

$$[d, e] \preceq [d', e'] \iff d \leq d' \text{ and } e \leq e'$$

Proposition 2.1 ($(ID, \preceq)$ is complete). *For any complete lattice $(D, \leq)$, the structure $(ID, \preceq)$ is a complete lattice. Furthermore for any subset $X \subseteq ID$, $X = \{[d_j, e_j] \mid j \in J\}$ not containing $[\top_{\leq}, \bot_{\leq}]$,*

$$\bigvee_{\preceq} X = [\vee_j d_j, \vee_j e_j] \text{ and } \bigwedge_{\preceq} X = [\wedge_j d_j, \wedge_j e_j]$$

An information ordering

Definition 2.3 ($\sqsubseteq$). *Define an ordering on ID by: for any $[d, e], [d', e'] \in ID$*

$$[d, e] \sqsubseteq [d', e'] \iff d \leq d' \text{ and } e' \leq e$$

We can prove the following theorem.

Theorem 2.1 ($(ID, \sqsubseteq)$ is complete). *For any complete lattice $(D, \leq)$, the structure $(ID, \sqsubseteq)$ is a complete lattice. Furthermore, for any collection $X = \{[d_j, e_j] \in ID \mid j \in J\}$ the $\sqsubseteq$-meet is given by the following*

$$\bigsqcap X = [\wedge_j d_j, \vee_j e_j]$$

The $\sqsubseteq$-join is given by:

$$\bigsqcup X = \bigcap X$$

Note that if this intersection is empty, then the join is $[\top_{\leq}, \bot_{\leq}]$. If the intersection is non-empty then the join is given by the interval $[\vee_j d_j, \wedge_j e_j]$.

2.2 Axiomatisation of trust structures

In this section we will consider trust structures as an abstract concept, attempting a formalisation of natural properties that these should satisfy. Until now, we have considered trust structures simply as triples $(X, \preceq, \sqsubseteq)$ where X is some set and $(X, \sqsubseteq)$ is either a complete lattice, requiring policies to be monotonic functions, or a cpo, requiring that policies be also continuous. However, one may wonder if there are additional requirements, e.g. relations between the two orderings, that are

[8] The intervals can be characterised as representatives of equivalence classes for a certain power domain construction.

naturally satisfied by the concrete structures we are interested in, and which can be justified also in an abstract framework. The goal of considering such additional requirements is at least two fold. Firstly, with more axioms satisfied, one may prove more properties of trust structures as an abstract concept, which are then satisfied in all applications. Secondly, thinking about such axioms may lead to a better understanding of the structures and applications. We choose the language of category theory for this formalisation. This provides a framework which has proven itself good at formalising general constructions such as the intervals. In the category theory framework one considers not just objects, e.g. the trust structures, but also relations between such objects, leading to a notion of *morphism of trust structures*.

We present now a preliminary attempt at such an axiomatisation. In choosing an axiomatisation, one should strive to achieve *few* simple and sufficient axioms. Our particular axioms have been chosen because they lead to a nice characterisation of the interval construction, they are satisfied by most of the examples we have considered, and they are supported by intuition. Consider the following categorical data, $\mathcal{T}$, which will be called the *category of trust structures*.

Definition 2.4 (Category of Trust Structures). *In the* category of trust structures*, $\mathcal{T}$,*

Objects *are quadruples, $T = (X, X_!, \preceq, \sqsubseteq)$, consisting of a set, X, a subset $X_! \subseteq X$ called the* dogmas *of T, and orderings $\preceq$, the trust ordering, and $\sqsubseteq$, the information ordering, of X, such that the following is satisfied: $(X, \preceq)$ is a partial order, $(X, \sqsubseteq)$ is a complete lattice and the subset, $X_! \subseteq X$, also called the* dogmatic points *of T, are such that $(X_!, \preceq_!)$, is a complete lattice, where $\preceq_!$ is the restriction of $\preceq$ to $X_!$. The following properties are satisfied:*

1. *($\bigsqcap$ is $\preceq$-monotonic) For any $x, y, z \in X$*

$$\text{if } x \preceq y \text{ then } x \sqcap_\sqsubseteq z \preceq y \sqcap_\sqsubseteq z$$

2. *(Information-Trust property) For any $x, y, z \in X$*

$$\text{if } x \preceq y \preceq z \text{ then } x \sqcap_\sqsubseteq z \sqsubseteq y$$

Morphisms *$f : S \to T$, between trust structures $S = (X, X_!, \preceq_X, \sqsubseteq_X)$ and $T = (Y, Y_!, \preceq_Y, \sqsubseteq_Y)$, are functions $f : X \to Y$ such that*

1. *(monotonic) f is monotonic with respect to both orderings $\sqsubseteq$ and $\preceq$.*
2. *(dogma) f preserves dogma, so that $f(d) \in Y_!$ whenever $d \in X_!$.*
3. *(strict) f is $\top_\sqsubseteq$-strict, and the restriction of f to dogmas, is strict, that is, preserves both top and bottom dogma.*

Identity and Composition *Identities 1_T are simply the identity functions, and composition is given by functional composition.*

Note 2.1. We write $\bigsqcup$ and $\bigsqcap$ for the $\sqsubseteq$ lub and glb respectively. When taking lub/glb in the lattice of dogmas we write $\bigvee_\preceq$ / $\bigwedge_\preceq$ to be explicit or simply $\bigvee$ / $\bigwedge$. Also, for any object $T = (X, X_!, \preceq_X, \sqsubseteq_X)$ the dogmatic points of T, $X_!$, is written both $X_!$ and $T_!$ when there can be no confusion.

We use here a paragraph, to give an informal "mental frame of reference" or intuition for understanding this category and the individual axioms. Consider a trust structure $T = (X, X_!, \preceq, \sqsubseteq)$. We think of the elements of X as "trust values", which are values that may not be entirely precise, and thus possibly contain some internal uncertainty. For $x, y \in X$, $x \sqsubseteq y$ reflects that the uncertainty in x is in

some sense greater than the uncertainty in y, that y is more precise than x, or that x can be refined into y. We can also have $x \preceq y$ which should be interpreted as x is less trust than y (whatever "trust" means in T). Some of these values are special in that they contain little or no internal uncertainty – this is the subset $X_!$ of "dogmas". A technical requirement is that the dogmas with the restriction of $\preceq$ is a complete lattice. Requiring that $(X, \sqsubseteq)$ is a complete lattice lets us define trust policies in any trust structure, as certain $\sqsubseteq$-monotonic functions. Because of the interpretation of the information order, the (binary) information meet $\sqcap$ can be interpreted as a function which constructs from a pair of values $x, y \in X$ an element $x \sqcap y$, which expresses uncertainty with respect to x and y, i.e. $x \sqcap y \sqsubseteq x, y$. Then the requirement that $\sqcap$ is $\preceq$-monotonic is natural, and the requirement that $(X, \sqsubseteq)$ be a complete lattice is equivalent to saying that for any subset $X_0 \subseteq X$ there is a unique $\sqsubseteq$-greatest element $\bigsqcap X_0$, with the property that it can be "refined into" any element in X_0. In addition to the "monotonicity" property, the "information-trust" property is a second simple axiom which chains together the two relations $\sqsubseteq$ and $\preceq$. Consider the following situation: we have $x \preceq y \preceq z$ – "spanning" a range of trust values. Then the element $x \sqcap z$ is an element expressing uncertainty between elements "in the range" from x to z, which should, intuitively, allow for refinement into y.

With respect to the morphisms we adopt also a minimalistic approach. We view morphisms as functions that preserve the structure of our trust structures. The monotonicity requirements are natural given that we are working with partial orders, and having introduced the notion of "dogma", it is only natural that morphisms preserve these. The requirement that morphisms be strict is a technical one, that we consider less fundamental.

We of course have:

Proposition 2.2. *$\mathcal{T}$ is a category.*

Example 2.5 *(Dogmatic structures).* Consider any "dogmatic" trust structure $T = (D, D, \leq, \leq)$, where $(D, \leq)$ is any complete lattice. The *dogmas* of T is then all of D (hence the name dogmatic). Since $\sqsubseteq = \preceq$, the information glb is equal to the $\leq$-glb, $\wedge_\leq$, which is obviously $\leq$-monotonic. To verify the information-trust property we simply note that if $x \leq y \leq z$ then $x \sqcap z = x \leq y$.

Example 2.6 *(mn-pairs).* Consider again the "*mn*-structure", T_{MN}, from Example 2.3. Take the set of dogmas $T_{MN!} = T_{MN} \setminus (\infty, \infty)$. It is not hard to verify that this is a complete lattice with $\preceq$, and that $\sqcap_\sqsubseteq$ is monotonic with respect to $\preceq$. The information-trust property also follows easily: if $m_0 \leq m_1 \leq m_2$ and $n_0 \geq n_1 \geq n_2$ then clearly

$$(m_0, n_0) \sqcap (m_2, n_2) = (m_0, n_2) \sqsubseteq (m_1, n_1)$$

Example 2.7 *(Convex sets).* Let $(D, \leq)$ be a complete lattice, and recall that a subset $X \subseteq D$ is *convex* if it satisfies the following property: for any $d \in D$ if there exists $x, y \in X$ so that $x \leq d \leq y$ then $d \in X$. Consider the power domain of convex subsets of D, $\mathcal{P}(D)$. Note that intervals are trivially convex, and so convex sets are, in a sense, a more general construction than the intervals. We choose the $\preceq$ order to be the Egli-Milner ordering, given by

$$X \preceq Y \iff \forall x \in X \exists y_x \in Y : x \leq y_x \text{ and } \forall y \in Y \exists x_y \in X : x_y \leq y$$

For the sake of this example, let us say that the property on the left of the "and" is the "Egli property", and that the right property is the "Milner property". For the information order $\sqsubseteq$ we choose the reverse subset ordering. Note that $(\mathcal{P}(D), \sqsubseteq)$ is then a complete lattice with the glb of any collection of sets $S = \{X_j \mid X_j \in$

$\mathcal{P}(D), j \in J\}$ given by the *convex closure* of the union of those sets, denoted

$$\bigsqcap S = \mathsf{cc}(\bigcup_{j \in J} X_j)$$

The dogmas $\mathcal{P}(D)_!$ are the singleton sets (which are trivially convex). Note that the restriction of the Egli-Milner ordering to dogmas gives a complete lattice isomorphic to $(D, \leq)$. *Monotonicity*: let $X, Y, Z \in \mathcal{P}(D)$ with $X \preceq Y$ and let $w \in X \sqcap Z$. Then there exists elements $w_0, w_1 \in X \cup Z$ so that $w_0 \leq w \leq w_1$. If w_1 in X then by $X \preceq Y$ and Egli, there exists $y_{w_1} \in Y$ with $w_1 \leq y_{w_1}$ by Egli, but then $w \leq y_{w_1}$ and we have Egli. If $w_1 \in Z$, then also $w_1 \in Y \sqcap Z$, and we trivially have Egli. Now for Milner: let $q \in Y \sqcap Z$, then there exists elements $q_0, q_1 \in Y \cup Z$ so that $q_0 \leq q \leq q_1$. If $q_0 \in Y$ then by $X \preceq Y$ and Milner, there exists $x_{q_0} \in X$ so that $x_{q_0} \leq q_0 \leq q$ which means Milner. If $q_0 \in Z$ then we trivially have Milner.
Information-trust property: let $X, Y, Z \in \mathcal{P}(D)$ with $X \preceq Y \preceq Z$, we want to show that $Y \subseteq X \sqcap Z$, so let $y \in Y$. By $X \preceq Y$ and Milner, there exists $x_y \in X$ with $x_y \leq y$. By $Y \preceq Z$ and Egli there exists $z_y \in Z$ with $y \leq z_y$. Thus we have $x_y \in X$ and $z_y \in Z$ with $x_y \leq y \leq z_y$. Then since $X \sqcap Z$ is the *convex* closure of $X \cup Z$, we have $y \in X \sqcap Z$.

Example 2.8 *(Evidence values).* Consider an extension of the T_{MN} structure. We consider instead *triples* of natural numbers, called evidence values. Similarly to Example 2.3, we think of such a triple $(s, i, c) \in \mathbb{N}^3$ as the result of making $s + i + c$ *observations* and classifying each observation as either *supporting* (s), *contradicting* (c) or *inconclusive* (i) for some hypothesis ϕ. Given this intuition we will consider the two orderings on evidence values.

Information order. The information ordering $\sqsubseteq$ of $\mathbb{N}^3$ is defined as follows:

$$(s, i, c) \sqsubseteq (s', i', c') \iff (s \leq s') \wedge (c \leq c') \wedge (s + i + c \leq s' + i' + c')$$

The rationale is that (s', i', c') represents more information than (s, i, c) if it can be obtained from (s, i, c) by performing some additional number of observations, or by refining the information about a particular observation (or both). By refining we mean to change an "inconclusive" to "supporting" or "contradicting".

Trust order. The trust ordering $\preceq$ of $\mathbb{N}^3$ is defined as:

$$(s, i, c) \preceq (s', i', c') \iff (s \leq s') \wedge (c \geq c') \wedge (s + i + c \leq s' + i' + c')$$

Here (s', i', c') expresses "more evidence in favour of ϕ" than (s, i, c) if it contains more supporting evidence, less contradicting evidence, and still at least as many observations. Intuitively one can obtain (s', i', c') from (s, i, c) by changing contradicting evidence to inconclusive or supporting, changing inconclusive to supporting, or by adding inconclusive or supporting events.

We will introduce some notation to simplify the rest of the exposition. For any $x = (s, i, c) \in \mathbb{N}^3$, write $x^+ = s$, $x^? = i$, $x^- = c$ and $\Sigma_x = s + i + c$. As in Example 2.3 we complete $\mathbb{N}^3$ by a $\preceq$-top $\top_\preceq$, a $\preceq$-bottom $\bot_\preceq$, and a $\sqsubseteq$-greatest element $\top_\sqsubseteq$. We write $\widehat{\mathbb{N}^3}$ for this completion, and extend the orderings as follows. For $\preceq$ we take $\top_\sqsubseteq$ unrelated to any $x \in \mathbb{N}^3$. For $\sqsubseteq$ we take $\bot_\preceq$ and $\top_\preceq$ mutually unrelated, and unrelated to any $(s, i, c) \neq \bot_\sqsubseteq$. We have that $(\widehat{\mathbb{N}^3}, \sqsubseteq)$ is a complete lattice: consider any subset $X \subseteq \widehat{\mathbb{N}^3}$. For the purpose of characterising the $\sqsubseteq$-meet, we can assume without loss of generality that X is not a finite chain, and that X doesn't contain $\top_\sqsubseteq$. If X doesn't contain $\bot_\preceq$ or $\top_\preceq$ then write $X = \{x_j \in \mathbb{N}^3 \mid j \in J\}$.

The $\sqsubseteq$-meet is given by $\bigsqcap X = \bar{x}$ where $\bar{x}^+ = \min_{j\in J} x_j^+$, $\bar{x}^- = \min_{j\in J} x_j^-$ and $\bar{x}^?$ is given by

$$\bar{x}^? = \max\{i \in \mathbb{N} \mid \forall j \in J : \bar{x}^+ + \ i \ + \ \bar{x}^- \leq \Sigma_{x_j}\}$$

If X contains one of $\bot_\preceq$ or $\top_\preceq$ then $\bigsqcap X = (0,0,0) = \bot_\sqsubseteq$. Note that in the former case, an algebraic expression for $\bar{x}^?$ is

$$\bar{x}^? = \min_{j\in J} \Sigma_{x_j} - \ \bar{x}^+ - \ \bar{x}^-$$

Take the dogmas $\widehat{\mathbb{N}^3}_!$ to be the triples $(m,0,n)$ together with $\top_\preceq$ and $\bot_\preceq$. Then the restriction of $\preceq$ to dogmas is isomorphic to the original T_{MN}-structure with its $\preceq$ ordering from Example 2.3, which is a complete lattice.

Monotonicity: In fact this property does *not* hold for a few elements, however, we conjecture that a less ad-hoc completion of the evidence domain would satisfy the monotonicity property for all elements. Let $x_0, x_1, z \in \widehat{\mathbb{N}^3}$ with $x_0 \preceq x_1$. Assume these are all in $\mathbb{N}^3$ – it is with some of the artificially added elements that the property fails. For $\mathbb{N}^3$: we then have $x_0^+ \leq x_1^+$, $x_0^- \geq x_1^-$ and $\Sigma_{x_0} \leq \Sigma_{x_1}$. Let $x_j \sqcap z = (\bar{s}_j, \bar{i}_j, \bar{c}_j)$. We have $\bar{s}_0 \leq \bar{s}_1$ and $\bar{c}_0 \geq \bar{c}_1$. Since $\Sigma_{x_0} \leq \Sigma_{x_1}$ clearly also $\min\{\Sigma_{x_0}, \Sigma_z\} \leq \min\{\Sigma_{x_1}, \Sigma_z\}$, which implies that $\Sigma_{x_0 \sqcap z} \leq \Sigma_{x_1 \sqcap z}$.

Information-Trust: Let $x, y, z \in \widehat{\mathbb{N}^3}$ with $x \preceq y \preceq z$ and assume again that all values are in $\mathbb{N}^3$. We have that $(x \sqcap z)^+ = x^+ \leq y^+$ and similarly $(x \sqcap z)^- = z^- \leq y$. Since $x \sqcap z \sqsubseteq x$ then we have $\Sigma_{x \sqcap z} \leq \Sigma_x \leq \Sigma_y$.

End of example

In the following section we shall examine the interval-constructed structures.

Intervals revisited

Consider again the interval construction from section 2.1. In this section we will briefly state the result that the interval construction can be viewed as part of an adjunction between $\mathcal{T}$ and a category of complete lattices.

Proposition 2.3. *The interval construction extends to a functor from the category of complete lattices with strict monotonic functions $\mathcal{CL}at$, to the category of trust-structures, $I : \mathcal{CL}at \to \mathcal{T}$, by:*

- $I(D, \leq) = (ID, ID_!, \preceq_{ID}, \sqsubseteq_{ID})$
- *for any strict monotonic function, $h : L \to K$ define a morphism of trust structures $I(h) : IL \to IK$ by: for any $[d,e] \in IL$*

$$Ih([d,e]) = [h(d), h(e)]$$

For any trust domain $T = (X, X_!, \preceq, \sqsubseteq)$ we have a forgetful mapping $! : \mathcal{T} \to \mathcal{CL}at$, which maps $!(T) = (X_!, \preceq_!)$. Since any morphism of trust structures restricts to a strict monotonic function on the dogma, we have

Proposition 2.4. *Let $!$ be given by $!(S) = (X_!, \preceq_!)$, for any trust structure $S = (X, X_!, \preceq, \sqsubseteq)$, and $!(f) = f_{|S_!}$, that is, the restriction of $f : S \to T$ to the dogmas. Then $!$ is a functor $! : \mathcal{T} \to \mathcal{CL}at$.*

Theorem 2.2. *The functor $I : \mathcal{CL}at \to \mathcal{T}$ is part of an adjunction $I \dashv !$. Furthermore the unit η of the adjunction is an isomorphism which makes I a co-reflector.*

3 The SECURE model for trust

The SECURE project [16, 17, 15] aims at providing a framework for decision-making for applications in GC environments, based on the notion of trust. The formal model for trust deployed is an instance of the framework from Section 2. In SECURE, each principal P has its own decision-making framework which is invoked when an application needs to make some decision involving another principal. The decision-making framework contains three primary components: the risk engine, the trust engine, and the collaboration monitor. At the most abstract level, the decision-making procedure works as follows. The collaboration monitor records the behaviour of principals with which P has interacted. This information together with a trust policy defines how P assigns trust values to any other principal. The trust information, in turn, serves as a basis for a risk analysis of any interaction. More specifically, with each type of interaction with a principal, say Q, there is a finite set of possible *outcomes* of the interaction, and the outcome that occurs is determined by the behaviour of Q. Each of these outcomes has an associated cost[9] which can be represented simply as a number, but can also be a more complex object like a probability distribution on some basic set of "cost values". Since the outcome depends on Q, the decision of how to interact is based on the trust in Q. In this set-up it is necessary that the trust value for Q carries enough information that estimation of the likelihood of each of the outcomes is possible. If this estimation is possible, one may start reasoning about risk, e.g. the *expected* or *maximal* cost of an interaction. Thus, the SECURE architecture brings forward the need for a formal model for trust supporting the approximation of likelihood of interaction outcomes, based on previous observations. We describe the trust model in SECURE, which supports this reasoning. We will use the mathematical structures known as event structures (see Nielsen, Winskel and Plotkin [20] for an original reference, and Winskel and Nielsen's handbook chapter [14] for an extensive reference).

Definition 3.1 (Event Structure). *An* event structure *is a triple* $(E, \leq, \#)$ *consisting of a set* E *of* events *which are partially ordered by* $\leq$*, the* causal dependency relation *(or causality relation), and* $\#$ *is a binary, symmetric, irreflexive relation* $\# \subset E \times E$*, called the* conflict relation*. The relations satisfy*

$$\textit{for any } e \in E\textit{, the set } \{e' \in E \mid e' \leq e\} \textit{ is finite,}$$

$$\textit{for any } e, e', e'' \in E\textit{, if } e \# e' \textit{ and } e' \leq e'' \textit{ then } e \# e''$$

We say that two events are independent *if they are not in either of the two relations.*

We will model the recording of a behavioural action as an event in an event structure. During an interaction, each party will obtain pieces of information, and these pieces are modelled by events. After some time each party will have observed some set of events occurring, which will constitute a special type of subset of events, called a configuration.

As an example, the event structure in Figure 1 could model a small scenario where a principal may ask a bank for the transfer of electronic cash from its bank account to an electronic wallet. After making the request, the principal observes that the request is either rejected or granted. After a successful transaction, the principal could observe that the cash sent in the transaction is forged or perhaps run an authentication algorithm to establish that it is authentic. Also, the principal could observe a withdrawal from its bank account with the present transaction's id, and this withdrawal may or may not be of the correct amount. The two basic relations on event structures have an intuitive meaning in our set up. An event may

[9] The term cost should be understood more generally as cost or *benefit*. If costs are represented as non-negative numbers, one might represent benefit as negative number.

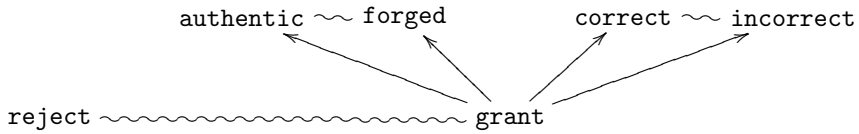

Figure 1: An event structure describing our example. The curly lines $\sim$ describe the immediate conflict relation and pointed arrows, the causal dependency relation.

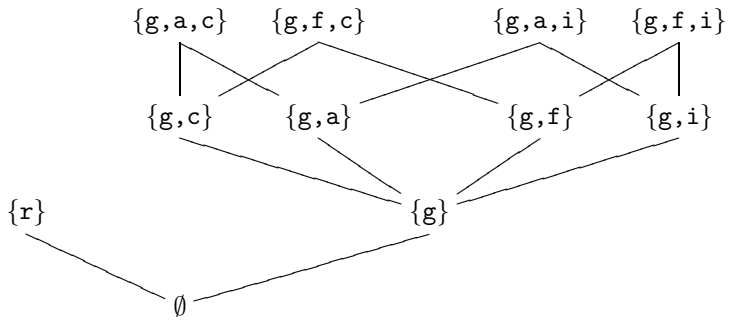

Figure 2: Configurations of the event structure in Figure 1. The lines indicate inclusion and the events are abbreviated.

exclude the possibility of the occurrence of a number of other events. In our example the occurrence of the event 'transaction rejected' clearly excludes the event 'transaction granted'. The causal dependency relation is also natural: some events are *only possible* when others have already occurred. In the example structure, 'money forged' only makes sense in a transaction where the transfer of money actually did occur. Whether the e-cash is forged and whether the correct amount is charged are two independent observations that may be observed, in any order, which is modelled as independence in the event structure.

Definition 3.2 (Configurations of an Event Structure). *Let $ES = (E, \leq, \#)$ be an event structure. Say that a subset of events $x \subseteq E$ is* consistent *if it satisfies the following two properties:*

1. *Conflict free: for any $e, e' \in x : e \not\# e'$ (i.e. $(e, e') \notin \#$).*
2. *Causally closed: for any $e \in x, e' \in E : e' \leq e \Rightarrow e' \in x$.*

Define the configurations *of ES, written $\mathcal{C}_{ES}$, to be the set of consistent subsets of E. We will define $\mathcal{C}^0_{ES}$ to be the* finite *configurations. Define relation $\rightarrow$ on $\mathcal{C}_{ES} \times E \times \mathcal{C}_{ES}$ by*

$$x \xrightarrow{e} x' \iff e \notin x \text{ and } x' = x \cup \{e\}$$

A (finite) configuration models information regarding the result of one interaction. Note that the *outcomes* of an action corresponds to the maximal configurations (ordered by inclusion) of the event structures, and then having complete knowledge of the outcome corresponds to having stored a maximal configuration as the information about the interaction. The configurations of our example is given in Figure 2.

We can now be more precise about the role of the collaboration monitor in the SECURE framework. Informally, its function is to record the behaviour of principals with whom interaction is made. For a particular type of interaction, the possible

events that may occur are modelled by an event structure, say ES. The information about the outcome of an interaction is then simply a configuration, $x \in \mathcal{C}^0_{ES}$.

Definition 3.3 (Interaction History). *Let $ES = (E, \leq, \#)$ be an event structure. Define an* interaction history *in ES to be a finite ordered sequence of configurations, $H = x_1 x_2 \cdots x_n \in {\mathcal{C}^0_{ES}}^*$. The individual components x_i in the history H will be called* interactions.

An interaction history in the event structure from Figure 1 could be the sequence $\{\mathtt{g}, \mathtt{a}, \mathtt{c}\}\{\mathtt{g}, \mathtt{c}\}\{\mathtt{g}\}\{\mathtt{r}\}$. The concept of interaction histories models one principal's recording of previous interactions with another. When the collaboration monitor learns about the occurrence of an event, e, this information is increased. We define a simple relation expressing this operation.

Definition 3.4 (Information Relation). *Let $ES = (E, \leq, \#)$ be an event structure and let $H = x_1 \cdots x_n$ and $K = y_1 \cdots y_n$ be interaction histories in ES, $e \in E$ an event, and $i \in \mathbb{N}, 1 \leq i \leq n$ an index. Define:*

$$H \overset{(e,i)}{\Rightarrow} K \iff x_i \overset{e}{\rightarrow} y_i \text{ and } \forall (1 \leq j \leq n) : j \neq i \Rightarrow x_j = y_j$$

and also let $\mathtt{new}$ be a special event $\mathtt{new} \notin E$ then

$$H \overset{\mathtt{new}}{\Rightarrow} H \cdot \emptyset$$

Let $H \Rightarrow K$ denote that either $H \overset{\mathtt{new}}{\Rightarrow} K$ or there exists $e \in E$, $i \in \mathbb{N}$ so that $H \overset{(e,i)}{\Rightarrow} K$, and $\Rightarrow^*$ denote the reflexive and transitive closure of $\Rightarrow$.

3.1 Evaluating evidence

We will equate the notion of trust values with "evidence values". That is, values expressing evidence regarding a particular partial outcome (i.e. a configuration). We will consider the derivation of such values based on interaction histories.

Consider an event structure $ES = (E, \leq, \#)$. A trust value will be a function from $\mathcal{C}_{ES}$ into a domain of *evidence values*. The function applied to a configuration $x \in \mathcal{C}_{ES}$ is then a value reflecting the evidence about x. It will be natural to express these evidence values as triples of natural numbers $(s, i, c) \in \mathbb{N}^3$ as in Example 2.8. Recall that the interpretation is that out of $s + i + c$ interactions, s of these *support* the occurrence of configuration x, c of these *contradict* it, and i are *inconclusive* about x in the sense that they do not support or contradict it.

Definition 3.5. *Let $ES = (E, \leq, \#)$ be an event structure and let x be a configuration of ES. Define the* effect *of x as a function, $\mathbf{eff}_x : \mathcal{C}_{ES} \to \mathbb{N}^3$ by*

$$\mathbf{eff}_x(w) = \begin{cases} (1,0,0) & \text{if } w \subseteq x \\ (0,0,1) & \text{if } x \# w \text{ (i.e. } \exists e \in x, e' \in w : e \# e') \\ (0,1,0) & \text{otherwise} \end{cases}$$

Also for $(s,i,c), (s',i',c') \in \mathbb{N}^3$ define $(s,i,c) + (s',i',c') = (s+s', i+i', c+c')$.

The intuition behind the definition of $\mathbf{eff}_x$ is the following. Think of x as a configuration which has already been observed. We are now considering engaging in another interaction which will end up in some configuration. Thus, we would like to estimate the likelihood of ending up in a particular configuration w, given that the last interaction ended in x. There are exactly three cases for any configuration w: if $w \subseteq x$ then the fact that x occurred last time supports the occurrence of w. If instead $x \# w$ then x contains an event which rules out the configuration w.

Finally, if neither of these are the case, i.e. w didn't occur but also wasn't excluded, we say that x is inconclusive about w. There is a strong similarity between this division of configurations in three disjoint classes and the way Jøsang [23] derives his uncertain probabilities in the Dempster-Shafer framework for evidence [21]. We return to this later.

Definition 3.6. *Let $ES = (E, \leq, \#)$ be an event structure, define the function* $\mathbf{eval} : {\mathcal{C}^0_{ES}}^* \to (\mathcal{C}_{ES} \to \mathbb{N}^3)$*:*

$$\mathbf{eval}(x_1 x_2 \cdots x_n) = \lambda w. \sum_{i=1}^{n} \mathbf{eff}_{x_i}(w)$$

We would like to note that the functions **eff** and **eval** allow for many useful variations when computing trust values from interaction histories. For example, suppose we want to model a "memory" so that a principal only remembers the last $M + 1 \in \mathbb{N}$ interactions. This could be done by simply taking

$$\mathbf{eval}^M(x_1 x_2 \cdots x_n) = \lambda w. \sum_{i=n-M}^{n} \mathbf{eff}_{x_i}(w) = \mathbf{eval}(x_{n-M} x_{n-M+1} \cdots x_n)$$

One could also imagine older interactions counting less, which could be modelled by scaling and rounding of the value of, say, the interactions older than a certain boundary.

Recall the two orderings on evidence values.

Information order. The information ordering $\sqsubseteq$ of $\mathbb{N}^3$ is defined as follows:

$$(s, i, c) \sqsubseteq (s', i', c') \iff (s \leq s') \wedge (c \leq c') \wedge (s + i + c \leq s' + i' + c')$$

Trust order. The trust ordering $\preceq$ of $\mathbb{N}^3$ is defined as:

$$(s, i, c) \preceq (s', i', c') \iff (s \leq s') \wedge (c \geq c') \wedge (s + i + c \leq s' + i' + c')$$

We restate the result from Example 2.8 as a Theorem, as it is important for section 3.2 on policy languages.

Theorem 3.1. *The structure $(\widehat{\mathbb{N}^3}, \sqsubseteq)$ is a complete lattice. For $(s_0, i_0, c_0), (s_1, i_1, c_1) \in \mathbb{N}^3$, the binary join is given by $(s_0, i_0, c_0) \sqcup (s_1, i_1, c_1) = (\bar{s}, \bar{i}, \bar{c})$ where $\bar{s} = \max\{s_0, s_1\}$, $\bar{c} = \max\{c_0, c_1\}$ and*

$$\bar{i} = \min\{i \in \mathbb{N} \mid \bar{s} + i + \bar{c} \geq \max\{s_0 + i_0 + c_0, s_1 + i_1 + c_1\}\}$$

Furthermore, the structure $(\mathbb{N}^3, \preceq)$ is a lattice. The binary $\preceq$-join is given by $(s_0, i_0, c_0) \vee (s_1, i_1, c_1) = (\hat{s}, \hat{i}, \hat{c})$ where $\hat{s} = \max\{s_0, s_1\}$, $\hat{c} = \min\{c_0, c_1\}$ and

$$\hat{i} = \min\{i \in \mathbb{N} \mid \hat{s} + i + \hat{c} \geq \max\{s_0 + i_0 + c_0, s_1 + i_1 + c_1\}\}$$

The meet is obtained dually. Finally, the binary join and meet functions for the trust order, $\vee, \wedge : \mathbb{N}^3 \times \mathbb{N}^3 \to \mathbb{N}^3$ are monotonic with respect to the information order.

In the following we use $\sqsubseteq$ also for the pointwise extension of $\sqsubseteq$ to trust values, i.e. the functions $\mathcal{C}_{ES} \to \widehat{\mathbb{N}^3}$. We can relate the relation $\Rightarrow^*$ on interaction histories with the information relation on trust values.

Proposition 3.1. *Let ES be an event structure. The function* $\mathbf{eval} : {\mathcal{C}^0_{ES}}^* \to \mathcal{C}_{ES} \to \mathbb{N}^3$ *is monotonic in the sense that for all interaction histories $H, K \in {\mathcal{C}^0_{ES}}^*$, if $H \Rightarrow^* K$ then* $\mathbf{eval}(H) \sqsubseteq \mathbf{eval}(K)$.

$$
\begin{array}{rll}
\pi ::= & \star : \tau & \text{(default policy)} \\
| & p : \tau ; \pi & (p \in \mathcal{P}\text{, specific policies}) \\
\tau ::= & p ?_{loc} q & \text{(local reference to } p, q \in \mathcal{P} \cup \{\star\}) \\
| & p ? q & \text{(policy reference to } p, q \in \mathcal{P} \cup \{\star\}) \\
| & \tau_1 \ \mathsf{binop} \ \tau_2 & \text{(binary operation } \mathsf{binop} \in \{\wedge, \vee, \sqcap, \sqcup\})
\end{array}
$$

Figure 3: An example policy language.

Some information is discarded by **eval**, and the following proposition explains what is lost. The function **eval** is injective up to rearranging the order of interactions.

Proposition 3.2. *Let $ES = (E, \leq, \#)$ be an event structure and $H, K \in \mathcal{C}^0_{ES}$ be configurations, $H = x_1 x_2 \cdots x_n$ and $K = y_1 y_2 \cdots y_m$. If $\mathbf{eval}(H) = \mathbf{eval}(K)$ then $m = n$ and there exists a permutation on n elements $\sigma : [n] \xrightarrow{\sim} [n]$ so that*

$$H = \sigma(K) \overset{(def)}{=} y_{\sigma(1)} y_{\sigma(2)} \cdots y_{\sigma(n)}$$

Returning to the SECURE architecture, the risk engine uses trust values to derive estimates on the likelihood of the various outcomes. Our trust values convey sufficient information to enable estimation of probability distributions on the configurations. There are several ways to do this, depending on the application. For example one might derive an opinion $\omega_x = (b_x, u_x, d_x)$ for $x \in \mathcal{C}_{ES}$ in the sense of Jøsang, which gives rise to a probability pdf [23, 33].

3.2 Policies

As mentioned, the trust model in the SECURE project is an instance of the framework in Section 2. We will extend the fixed-point framework with a notion of *local* information. This is simply trust information that a principal p stores, which is derived from the *direct interaction* that p has had with other principals. Recall that in the framework, each principal defines a local *trust policy* which specifies its trust. We give an example of a language for writing trust policies.

The syntax of our simple language is given in Figure 3. A policy is a list of specific policies, terminated by a general policy. The specific policies explicitly name a principal and a corresponding trust expression (τ), whereas the general policy applies to any principal not explicitly listed. In this simple example language, the trust expressions are built up from the basic constructs of "local reference" and "policy reference", and these can then be combined with the various joins and meets we have available. The two types of references are similar in that both refer to a principal p's trust value for a principal q. The difference is that the local reference refers to the trust value derived from p's *personal interactions* with q, whereas the policy reference instead refers to the value that p would compute for q using its *policy*.

The semantics of a policy is an information monotonic function as in Section 2. However, we extend the framework slightly by interpreting these functions relative to an environment providing for each pair (p, q) of principals, a trust value, which we think of as being p's interaction history with q, evaluated as specified in the previous section. This serves as the data for the local references. Let $obs : \mathcal{P} \to \mathcal{P} \to \mathcal{C}_{ES} \to \mathbb{N}^3$ be a fixed function representing the local data.

The semantics of a policy π is a function which takes as input the observation data *obs*, and gives as output a $\sqsubseteq$-monotonic function mapping the global current

$$\begin{array}{rcl} [\![\star : \tau]\!]^{obs} & = & \lambda m \in GS.\lambda y \in \mathcal{P}.[\![\tau]\!]^{obs}(m)([\star \mapsto y]) \\ [\![p : \tau\ ; \pi]\!]^{obs} & = & \lambda m \in GS.\lambda x \in \mathcal{P}.\ \texttt{if}\ (x = p)\ \texttt{then}\ [\![\tau]\!]^{obs}(m)([\star \mapsto p]) \\ & & \qquad\qquad \texttt{else}\ [\![\pi]\!]^{obs}(m)(x) \end{array}$$

Figure 4: Semantics of the policy language: syntactic category π

$$\begin{array}{rl} [\![Y ?_{loc} Z]\!]^{obs}(m)(env) = obs\ (env^{\dagger}\ Y)\ (env^{\dagger}\ Z) & (\text{for } Y, Z \in \mathcal{P} \cup \{\star\}) \\ [\![Y ? Z]\!]^{obs}(m)(env) = m\ (env^{\dagger}\ Y)\ (env^{\dagger}\ Z) & (\text{for } Y, Z \in \mathcal{P} \cup \{\star\}) \\ [\![\tau_1\ \mathsf{binop}\ \tau_2]\!]^{obs}(m)(env) = \big([\![\tau_1]\!]^{obs}(m)(env)\big)\ [\![\mathsf{binop}]\!]\ \big([\![\tau_2]\!]^{obs}(m)(env)\big) & \end{array}$$

Figure 5: Semantics of the policy language: syntactic category τ

trust state (an element in $GS = \mathcal{P} \to \mathcal{P} \to (\mathcal{C}_{ES} \to \widehat{\mathbb{N}^3})$), to a local trust state (an element of $LS = \mathcal{P} \to (\mathcal{C}_{ES} \to \widehat{\mathbb{N}^3})$). We denote this as

$$[\![\pi]\!]^{obs} : GS \to LS$$

The semantic function $[\![\cdot]\!]^{obs}$ is defined by structural induction on the syntax of π in Figure 4. The definitions make use of the semantic function in Figure 5, which essentially maps the syntactic category τ to an element of $\mathcal{C}_{ES} \to \widehat{\mathbb{N}^3}$. This is interpreted relative to observations *obs* and the current trust state $m : GS$, but also relative to an environment, $env : \{\star\} \to \mathcal{P}$, which interprets $\star$ as a name in $\mathcal{P}$. The *env* function extends trivially to a function $env^{\dagger}$, of type $\{\star\} \cup \mathcal{P} \to \mathcal{P}$ (the identity on non-$\star$ elements). The semantics of a binop is the corresponding $\sqsubseteq$ or $\preceq$ lub/glb[10], which is $\sqsubseteq$-monotonic by Theorem 3.1. Similarly to Section 2, we can now view a collection of mutually referring policies, $\Pi^{obs} = ([\![\pi_P]\!]^{obs} \mid P \in \mathcal{P})$ as defining a "web of trust", and define the unique monotonic function Π^{obs}_{λ}

$$\Pi^{obs}_{\lambda} = \left\langle [\![\pi_P]\!]^{obs} : P \in \mathcal{P} \right\rangle : GS \to GS$$

We then define the unique global trust state, given that the local observations is as specified in $obs : \mathcal{P} \to \mathcal{P} \to \mathcal{C}_{ES} \to \mathbb{N}^3$, as $\mathsf{gts}(\Pi, obs) = \mathsf{lfp}\ \Pi^{obs}_{\lambda}$.

We give a couple of examples of event structures and policies.

Example 3.1 *(SECURE spam).* A prototype application for filtering spam has been developed using the SECURE framework [16]. Consider the simple event structure $ES = (E, \leq, \#)$ with two conflicting events: $E = \{\texttt{spam}, \texttt{notspam}\}$. A network of cooperating principals may then collaborate to identify spammers:

$$\begin{array}{rcl} \pi_A & = & \star : (A\ ?_{loc}\ \star) \sqcup (B\ ?\ \star) \\ \pi_B & = & \star : (B\ ?_{loc}\ \star) \sqcup ((A\ ?\ \star) \wedge (C\ ?\ \star)) \\ \pi_C & = & \star : (C\ ?_{loc}\ \star) \end{array}$$

Now if a spammer sends a mail to C which is classified as "spam", then C will assign the value $(1, 0, 0)$ to the configuration $\{\texttt{spam}\}$. Say that the spammer is tricky and sends to A a non-spam message. At some point later, the spammer returns to send a spam-message to A, which then computes its trust value. Locally it will obtain

[10] We use a strict version of the $\preceq$-lub/glb which is the $\top$-strict extension of $\vee, \wedge : \mathbb{N}^3 \times \mathbb{N}^3 \to \mathbb{N}^3$ to a function $\widehat{\mathbb{N}^3} \times \widehat{\mathbb{N}^3} \to \widehat{\mathbb{N}^3}$ which is also monotonic.

the value $(0, 0, 1)$ on the "spam" configuration, which indicates that no spam has been previously sent. However, since A depends on B, which depends on C, the final result for A will be $(1, 0, 1)$. Whether to allow the message or not is up to the local *security* policy of A.

Example 3.2 *(Beta).* The beta reputation system is a reputation-based rating system, developed by Jøsang [33]. Each binary interaction is rated by both participants as a pair of values (r, s) where "r reflects the degree of satisfaction and s reflects the degree of dissatisfaction". The β-reputation systems uses "continuous" values, but to fit with our model, we use instead natural numbers. Consider the event structure $ES = (E, \emptyset, \emptyset)$ with two independent events, $E = \{\texttt{satisfied}, \texttt{dissatisfied}\}$. Each interaction is then rated as $\{\texttt{satisfied}\}$ or $\{\texttt{dissatisfied}\}$, and since Jøsang allows an interaction to have both a degree of dissatisfaction and a degree of satisfaction, we allow also configuration $\{\texttt{satisfied}, \texttt{dissatisfied}\}$. At each interaction one has to scale the trust values (s, i, c) to get arbitrary "(r, s)" pairs with the appropriate ratio. All principals deploy the same "global" policy π, which computes the sum of all (sic)-triples:

$$\pi = \star : \sum_{p \in \mathcal{P}} (p \ ?_{loc} \ \star)$$

Note that one difference from the beta reputation system, is that Jøsang first converts ratings into opinions (b, u, d) before the global trust values are computed.

Example 3.3 *(P2P file-sharing).* Shmatikov and Talcot have presented a comprehensive framework for reputation systems [28]. We have not yet analysed the exact connection between our framework and theirs, but suspect we can encode at least part of their framework in ours. One of the example applications of their system is a reputation mechanism for P2P file-sharing. Consider an event structure with two conflicting events, $E = \{\texttt{ul}, \texttt{dl}\}$. In each interaction a principal either downloads a file $\{\texttt{dl}\}$ uploads a file $\{\texttt{ul}\}$. In their application example a principal's trust value depends *only* on the information available locally, i.e. p deploys policy

$$\pi_p = \star : p \ ?_{loc} \ \star$$

An example security policy[11] is then to say that the number of uploads must be at least a third of the number of downloads, a function which is easily computed from our trust values.

3.3 Transfer of information

The example policy language in the previous section allows principals to share trust information by means of the reference constructs. However, we were implicitly assuming that all principals agree on the event structure used. One event structure describes a particular context, i.e. there is one event structure for each possible way of interacting. It is useful to be able to map trust values between contexts that are somehow related, e.g. if one has only very little information about context ES_1 but much information about a related context ES_2, it is often useful to somehow apply the knowledge of ES_2 to give an estimate in ES_1. We propose a definition of a morphism of event structures enabling such an information transfer.

Definition 3.7 (Morphism of event structures). *Let $ES = (E, \leq, \#)$ and $ES' = (E', \leq', \#')$ be event structures. A* morphism of event structure*, $\eta : ES \to ES'$ is a function $\eta : E' \to \mathbf{2}^E$ which has the following two properties:*

[11] Their framework deals also with specification of decision-making or security policy. We distinguish the problem of specifying trust values, i.e. specification of *trust policy*, from the problem of implementing a security *decision* based on a trust value, i.e. specification of *security policy*.

1. *Monotonic: For any* $e', e'' \in E'$

$$e' \leq' e'' \;\Rightarrow\; \forall e_2 \in \eta(e'')\exists e_1 \in \eta(e') : e_1 \leq e_2$$

2. *Preserves conflict: For any* $e', e'' \in E'$

$$e' \#' e'' \;\Rightarrow\; \forall e_1 \in \eta(e')\forall e_2 \in \eta(e'') : e_1 \# e_2$$

A morphism $\eta : ES \to ES'$ can be thought of as a particular way of transferring information from ES to ES'. The idea is that $e \in \eta(e')$ means that an occurrence of e in ES is an indication of the event e' occurring in ES'. We will think of the set $\eta(e')$ as a disjunction of conditions in the sense that e' occurs if there is some $e \in \eta(e')$ which has occurred in ES. If $\eta(e') = \emptyset$ then we say that e' has no enabling condition under η.

Definition 3.8 (Category of Event Structures, E). *Consider the following categorical data, which we will call the category of event structures, and denote* **E**.

- *Objects are event structures* $ES = (E, \leq, \#)$
- *Morphisms* $\eta : ES \to ES'$*, are the morphisms of Definition 3.7.*
- *Identities* $1_{ES} : ES \to ES$ *are the functions* $1_{ES} : E \to \mathbf{2}^E$ *given by*

$$1_{ES}(e) = \{e\}$$

- *For* $\eta : ES \to ES'$ *and* $\epsilon : ES' \to ES''$ *composition,* $\epsilon \circ \eta : ES \to ES''$ *is given by the following function* $\epsilon \circ \eta : E'' \to \mathbf{2}^E$

$$\epsilon \circ \eta(e'') = \bigcup_{e' \in \epsilon(e'')} \eta(e')$$

Proposition 3.3 (E is a category). *The definition of* **E** *yields a category.*

A morphism, $\eta : ES \to ES'$ can then be used to map configurations of ES to configurations of ES' by the mapping,

$$\lambda x \in \mathcal{C}_{ES}.\{e' \in E' \mid \exists e \in \eta(e') : e \in x\}$$

The axioms of morphisms imply this set is always a configuration, and the fact that **E** constitutes a category means that we can compose the information transfer functions to obtain information transfer functions.

Generalised morphisms of event structures

We have just defined morphisms of event structures $\eta : ES \to ES'$ as functions $\eta : E' \to \mathbf{2}^E$ with the interpretation that event e' occurs in ES' if one of the prime configurations $\lceil e \rceil$ with $e \in \eta(e')$ occurs in ES. At the cost of more technicality, we can generalise this definition to allow *arbitrary* finite configurations instead of only prime configurations, thus obtaining a more expressive evidence transfer framework.

Definition 3.9 (Generalised morphism). *Let* $ES = (E, \leq, \#)$ *and* $ES' = (E', \leq', \#')$ *be event structures. A (generalised) morphism,* $\eta : ES \to ES'$ *is a function* $\eta : E' \to \mathbf{2}^{\mathcal{C}^0_{ES}}$ *which has the following two properties:*

1. *Monotonic: For any* $e', e'' \in E'$

$$e' \leq e'' \;\Rightarrow\; \forall x_2 \in \eta(e'')\exists x_1 \in \eta(e') : x_1 \subseteq x_2$$

2. Preserves conflict: For any $e', e'' \in E'$

$$e' \#' e'' \;\Rightarrow\; \forall x_1 \in \eta(e') \forall x_2 \in \eta(e'') : x_1 \# x_2$$

Note 3.1. If we define the pre-order $\preceq$ on $\mathbf{2}^{\mathcal{C}^0_{ES}}$ to be

$$X \preceq Y \iff \forall y \in Y \exists x \in X : x \subseteq y$$

then the monotonicity requirement amounts to saying that η be monotonic. Note also that this definition is the obvious generalisation of Definition 3.7 to arbitrary finite configurations.

We will still think of the set $\eta(e')$ as a disjunction of conditions in the sense that e' occurs if there is some $x \in \eta(e')$ which has occurred in ES. When $\eta(e')$ is non-empty we shall call it the enabling condition for e' under η. If $\eta(e')$ is empty then e' has no enabling condition under η. Note that if $X, Y \subseteq \mathcal{C}^0_{ES}$ then $X \preceq Y$ intuitively means that the requirement X is weaker than requirement Y: suppose some $y \in Y$ has occurred then there exists an $x \in X$ which has also occurred since $x \subseteq y$. We shall formalise this intuition some more. Let $z \in \mathcal{C}_{ES}$ and define a set

$$\texttt{enabled}(\eta, z) = \{e' \in E' \mid \exists y \in \eta(e') : y \subseteq z\}$$

Note that for events $e', e'' \in E'$, if $\eta(e') \preceq \eta(e'')$ then for *any* $z \in \mathcal{C}_{ES}$

$$e'' \in \texttt{enabled}(\eta, z) \Rightarrow e' \in \texttt{enabled}(\eta, z)$$

We introduce a bit of notation. Let $e' \in E'$ and assume $\eta(e') = \{x_i \in \mathcal{C}^0_{ES} \mid i \in I\}$. We shall sometimes write $\bigvee_{i \in I} x_i$ for such a set, so writing $\eta(e') = \bigvee_{i \in I} x_i$ is common – this suggests the understanding of the set as a disjunction. In the finite case, $\eta(e') = \{x_1, x_2, \ldots, x_n\}$ we sometimes write $\eta(e') = x_1 \vee x_2 \vee \cdots \vee x_n$ and if $n = 1$ simply $\eta(e') = x_1$.

Example 3.4 *(Rooted event sub-structures).* Consider an event structure $ES = (E, \leq, \#)$ and some fixed finite configuration $x \in \mathcal{C}^0_{ES}$. We will look at an event structure ES_x containing a special *event* x, for which the occurrence of x in ES_x will represent the occurrence all events of configuration x in ES. The other events in ES_x will be the subset of events in E that can occur after x has already occurred. This structure ES_x will be called the event sub-structure rooted at x. Formally, it is defined as follows: Let $ES_x = (E_x, \leq', \#')$ where $E_x = E \setminus (x \cup \#(x)) \cup \{x\}$. Here $\#(x)$ denotes the subset of events in E which are in conflict with some event in x. The relations $\leq'$ and $\#'$ are given by the following rules: $\leq'$ is the restriction of $\leq$ to E_x and we have for any $e_x \in E_x : x \leq e_x$. $\#'$ is simply the restriction of $\#$ to $E_x \setminus \{x\}$. This can be verified to be an event structure. Now define a morphism $\eta_x : ES \to ES_x$ as follows: η_x is the function of type $E_x \to \mathbf{2}^{\mathcal{C}^0_{ES}}$ given by

$$\eta = \begin{cases} x \mapsto x \\ e_x \mapsto x \cup \lceil e_x \rceil_{ES} & \text{for any } e_x \in E_x, e_x \neq x \end{cases}$$

Here $\lceil e_x \rceil_{ES}$ is the $\leq$-downwards closure in ES (and not ES_x). This is well-defined: since $e_x \in E_x$ is not in conflict with x this means that $x \cup \lceil e_x \rceil_{ES}$ is a finite configuration of ES. Also, η_x satisfies the conflict and monotonicity-properties.

Proposition 3.4. *Let* $ES = (E, \leq, \#)$ *and* $ES' = (E', \leq', \#')$ *be event structures and* $\eta : ES \to ES'$ *a morphism of event structures. Define a function* $\eta^* : \mathcal{C}_{ES} \to \mathcal{C}_{ES'}$ *by*

$$\eta^*(x) = \texttt{enabled}(\eta, x)$$

Then η^* *is well defined in the sense that for any* $x \in \mathcal{C}_{ES}$ *we have* $\eta^*(x) \in \mathcal{C}_{ES'}$. *Furthermore this function is monotonic.*

Example 3.5 *(example (3.4) continued).* Consider again the setting from example (3.4). Now let $y \in \mathcal{C}_{ES}$ suppose that $x \subseteq y$. Then

$$\begin{aligned} \texttt{enabled}(\eta_x, y) &= \{e_x \in E_x \mid \exists z \in \eta_x(e_x) : z \subseteq y\} \\ &= \{x\} \cup \{e_x \in E_x \setminus \{x\} \mid \lceil e_x \rceil_{ES} \subseteq y\} \\ &= \{x\} \cup (y \setminus x) \end{aligned}$$

On the other hand suppose that $x \not\subseteq y$ then

$$\texttt{enabled}(\eta_x, y) = \{e_x \in E_x \mid \eta_x(e_x) \subseteq y\} = \emptyset$$

So the function $\eta_x^* : \mathcal{C}_{ES} \to \mathcal{C}_{ES_x}$ implements the feature of "ignoring" the configurations that don't contain x. The other configurations will be mapped to the equivalent configuration in E_x.

We want to prove that the generalised morphisms constitute a category. In particular, we need a way of composing morphisms. While this can be done, it is not as obvious as with the non-generalised morphisms. To do this we need the following notion. Say that finite configurations $x, x' \in \mathcal{C}^0_{ES}$ of an event structure are compatible if $x \cup x'$ is a configuration. Define a partial function $\bigwedge$ on finite sets of finite configurations by

$$\bigwedge X = \begin{cases} \bigcup X & \text{if the elements of } X \text{ are all mutually compatible} \\ \star & \text{otherwise.} \end{cases}$$

Note that by definition if $\bigwedge X$ is defined then it is a finite configuration. The idea is that if all configurations in X are compatible, and we view each element $x \in X$ as a condition, then $\bigwedge X$ is itself a condition representing the conjunction of all those conditions in X.

Now, for any $\eta : ES \to ES'$ define an extended function $\bar{\eta} : \mathcal{C}^0_{ES'} \to \mathbf{2}^{\mathcal{C}^0_{ES}}$, expressing the enabling condition for any *finite configuration* under η. The idea is the following. Let $x = \{e'_1, e'_2, \ldots, e'_n\} \in \mathcal{C}^0_{ES'}$, then $\eta(e'_j) = \bigvee_{k \in I_j} y_j^k$ expresses that e'_j occurs in ES' if one of the configurations in the disjunction occurs in ES. So what should we require for configuration x to occur? It occurs if each event $e'_j \in x$ occurs, so for each way of selecting a $y_j^k \in \eta(e'_j)$ for each $j = 1, 2, \ldots, n$ the conjunction of these configurations are a sufficient enabling condition for the set $x = \{e'_1, e'_2, \ldots, e'_n\}$.

Assume that $\eta(e'_j) = \bigvee_{k \in I_j} y_j^k$ for index sets $I_1, I_2, \ldots I_j, \ldots I_n$, and where $y_j^k \in \mathcal{C}^0_{ES}$. Recall that the notation $\bigvee_{i \in I} x_i$ means the set $\{x_i \mid i \in I\}$. We will extend this notation to allow the x_i to be $\star$ in which case such an x_i is ignored. More formally, for $i \in I$, let $x_i \in \mathcal{C}^0_{ES} \cup \{\star\}$ and define $\bigvee^\star_{i \in I} x_i$ to mean the set $\{x_i \mid i \in I, x_i \neq \star\}$. Finally we can define $\bar{\eta}$ by:

$$\bar{\eta}(x) = \bigvee^\star_{\sigma \in \prod_{j=1}^n I_j} \left(\bigwedge_{j=1}^n y_j^{\sigma(j)} \right)$$

Definition 3.10 (Category of Generalised Event Structures, GE). *Consider the following categorical data:*

- *Objects are event structures* $ES = (E, \leq, \#)$
- *Morphisms* $\eta : ES \to ES'$*, are the morphisms of definition 3.9.*
- *Identities* $1_{ES} : ES \to ES$ *are the functions* $1_{ES} : E \to \mathbf{2}^{\mathcal{C}^0_{ES}}$ *given by*

$$1_{ES}(e) = \lceil e \rceil$$

- *For $\eta : ES \to ES'$ and $\epsilon : ES' \to ES''$, composition $\epsilon \circ \eta : ES \to ES''$ is given by the following function $\epsilon \circ \eta : E'' \to \mathbf{2}^{\mathcal{C}^0_{ES}}$*

$$\epsilon \circ \eta(e'') = \bigcup_{x' \in \epsilon(e'')} \bar{\eta}(x')$$

Proposition 3.5 (GE is a category). *The definition of* **GE** *yields a category.*

3.4 A connection

We will consider in more depth a connection between the role of event structures and the use of the Dempster-Shafer theory of evidence [21] in the subjective logic of Jøsang [23]. The basis of the subjective logic is the notion of a frame of discernment. A frame of discernment is given by a set $\theta = \{p_j \mid j \in J\}$ of atomic propositions, for which exactly one is true at any time. One then considers the power set $\mathbf{2}^\theta$ and assumes a so-called belief mass assignment (BMA), $m : \mathbf{2}^\theta \to [0,1]$ with the property that $m(\emptyset) = 0$ and $\sum_{x \in \mathbf{2}^\theta} m(x) = 1$. A BMA indicates the belief that an entity assigns to each of the sets $x \in \mathbf{2}^\theta$. Belief assigned to a set x is interpreted as belief that one of the propositions in x is true.

Jøsang does not treat how one might obtain such belief mass assignments, but some of his examples suggest that one way is to consider a number of experiments, each with an outcome $x \in \mathbf{2}^\theta$, and somehow have this lead to a BMA.

Given a BMA, $m : \mathbf{2}^\theta \to [0,1]$, Jøsang derives his three functions $b, u, d : \mathbf{2}^\theta \to [0,1]$ given by

$$\begin{aligned} b(x) &= \sum_{y \subseteq x} m(y) \\ d(x) &= \sum_{y \cap x = \emptyset} m(y) \\ u(x) &= \sum_{y \not\subseteq x,\ y \cap x \neq \emptyset} m(y) \end{aligned}$$

which satisfy $b + u + d = 1$. His subjective logic is then developed for such opinion triples.

The notion of event structures generalises the concept of frames of discernment. We first consider a slight generalisation of event structures. Say that a conflict-generalised event structure is a triple $(E, \leq, \#)$ where $\leq$ is the causal dependency relation as in ordinary event structures, but the conflict relation is no longer binary. Instead, let $\# \subseteq E \times \mathbf{2}^E$. This should be understood as follows: an event $e \in E$ is in conflict with the set $x \subseteq E$ if $(e, x) \in \#$, and so if x has occurred, then this excludes the event e from occurring. This generalises ordinary event structures, in that for ordinary structures, the sets x must always be singletons. The relations satisfy a generalisation of the axioms of event structures, which we do not go into detail with here. Now given any finite frame of discernment $\theta = \{p_1, p_2, \ldots, p_n\}$, define a conflict-generalised event structure, $ES_\theta = (E, \emptyset, \#)$, with $E = \{\bar{p}_i \mid i \in 1, 2, \ldots n\}$. One should understand the event $\bar{p}_i$ as learning that p_i is impossible. The conflict relation is given by

$$(e, x) \in \# \iff x \cup e = E$$

Let us consider now the configurations of this event structure. Since the causal dependency relation is empty, the configurations are exactly the conflict-free subsets of events. This means that we get all subsets of E, except for E itself. Now, consider the correspondence between non-empty subsets of θ and configurations of

ES_θ, $\phi : 2^\theta \setminus \{\emptyset\} \to \mathcal{C}_{ES_\theta}$ given by: $\phi(x) = \{\bar{p} \mid p \in \theta, p \notin x\}$. It is not hard to see that this is a bijection, and that for any non-empty $x, y \in \mathbf{2}^\theta$ we have $x \subseteq y$ iff $\phi(x) \supseteq \phi(y)$. Furthermore, $x \cap y = \emptyset$ iff $\phi(x) \cup \phi(y) = E$ iff $\phi(x) \# \phi(y)$. This means that our way of deriving (*sic*)-triples is the natural generalisation of the Jøsang-Dempster-Shafer technique of deriving "opinions" or "belief and plausibility"-functions on frames of discernment.

4 Distributed computation of least fixed points

We have seen in the previous sections that a collection of monotone policies $\Pi = (\pi_p \mid p \in \mathcal{P})$ defines a unique *global trust state*, $\mathsf{gts} = \mathsf{lfp}\ \Pi_\lambda : \mathcal{P} \to \mathcal{P} \to T$, defining p's trust in q as $\mathsf{gts}(p)(q)$. In order to make security decisions, each principal p will need to reason about its trust in others, that is, the values of $\mathsf{gts}(p)$. When the lattice $(T, \sqsubseteq)$ is of finite height h, the lattice $(\mathcal{P} \to \mathcal{P} \to T, \sqsubseteq)$ has height $|\mathcal{P}|^2 \cdot h$, and the least fixed point can, in principle, be computed by finding the first identity in the chain $(\lambda p.\lambda q.\bot_\sqsubseteq) \sqsubseteq \Pi_\lambda(\lambda p.\lambda q.\bot_\sqsubseteq) \sqsubseteq \Pi_\lambda^2(\lambda p.\lambda q.\bot_\sqsubseteq) \sqsubseteq \ldots \sqsubseteq \Pi_\lambda^{|\mathcal{P}|^2 \cdot h}(\lambda p.\lambda q.\bot_\sqsubseteq)$. However, in the environment we envision, such a computation is infeasible: the number of principals $|\mathcal{P}|$, though finite, will be *very* large, and even if resources were available we can not assume any central authority present to make such a computation.

One crucial observation is that for p to compute $\mathsf{gts}(p)$, it may not be necessary to compute the *global* value $\mathsf{lfp}\ \Pi_\lambda$, and *then* apply this function to p. Instead, as has been recognised also by Vergauwen *et al.* [38], it is often possible to compute a so-called *local fixed point value* directly, an observation which applies even more in our scenario. The reason for this is that even though, in principle, the policies $\pi : (\mathcal{P} \to \mathcal{P} \to T) \to \mathcal{P} \to T$ can depend on the values for *all* principals, this may not always be the case. The hypothesis we hope will hold in practice is that for a fixed principal p, then the set of principals that π_p *actually* depends on is a significantly smaller subset of $\mathcal{P}$, and so we can cut off a large number of principals that do not need to be involved in a computation of $\mathsf{gts}(p)$.

We will think of this in a slightly more abstract setting. We are given a lattice $(T, \sqsubseteq)$ of finite height[12] h, and a natural number $m \in \mathbb{N}$, which we think of as being large. We write $[m]$ for the set $\{1, 2, \ldots, m\}$. We have also a collection of m monotone functions $C = (f_i : i \in [m])$ of type $f_i : ([m] \to T) \to T$. These functions induce a unique global monotone function $F = \langle f_i : i \in [m] \rangle : ([m] \to T) \to [m] \to T$ with the property that $\mathsf{Proj}_i \circ F = f_i$ for all $i \in m$. F then has a unique least fixed point, $\mathsf{lfp}\ F \in [m] \to T$. We can define a dependency graph $G = ([m], D)$, where $[m]$ is the set of nodes, and the edges $D : [m] \to \mathbf{2}^{[m]}$ model (an over approximation of) the dependencies of the functions in C, i.e. we have $j \notin D(i)$ implies that function f_i does *not* depend on the value of "variable" j. We consider the nodes $[m]$ as network nodes that have memory and computational power, and which communicate with each other by means of message passing. We assume that each node knows all nodes that it depends on, i.e. node i knows all edges $D(i)$. Fix a particular node $R \in [m]$ which is called the *root*. The goal of the distributed algorithms in this section is for the root node to compute its *local fixed point value*: $\mathsf{lfp}\ F_R$, that is, the value $(\mathsf{lfp}\ F)(R)$. Note that one trivial "distributed" algorithm for computing this is to send all the functions to the designated node R, and then let R compute the sequence $\bot, F(\bot), F^2(\bot), \ldots$. This may be acceptable in some scenarios, but there are some issues to consider. Firstly, nodes $[m]$ may not be willing to reveal their entire functions f_i. Secondly, we are not exploiting

[12] This simplifies the exposition, T will be $\mathcal{P} \to T$ in trust computation applications, and then the height is $|\mathcal{P}| \cdot h$

the potential parallelisation of the computation which may give a significant speed up and distribute the computational burdens. Third, the encoding of a general function of type $T^m \to T$ is $\theta(|T|^m \log_2 |T|)$ bits.

In this section we describe algorithms for the computation of the local fixed point value for a fixed *root* node, R. We describe algorithms which are *globally synchronous* and *totally asynchronous* in the classification of Bertsekas ([34], pp. 91-92). There are many issues to consider in general when doing distributed computations, and the applications we have in mind introduce even more. We list some issues here, and return to these points in the last sub-section.

Robustness What happens if a node fails or, simply, is not willing to participate in a fixed point computation? Must the entire computation be aborted, or is partial recovery feasible?

Complexity and scalability Principals have varying computational resources, communication capabilities, and memory. The computational and communication requirements incurred by a fixed point computation must be as low as possible. Also, because of the large number of principals envisioned, massive scalability is required. This also makes it infeasible that everyone be involved in a fixed point computation.

Security Obviously principals have an interest in affecting the trust values that others assign to them. Can we ensure that an adversarial principal or party of colluding principals cannot affect trust values arbitrarily?

Privacy Principals may not be willing to reveal certain information about their trust policies.

Dynamics At any time every principal p has a trust policy. However, as time evolves p may obtain new information, e.g. by making new observations, which may lead to a change in policy. If such dynamic policy changes occurs during a fixed point computation, then this computation might be invalidated. Also, related to the area of *dynamic algorithms*, suppose that p's trust value for a principal has been computed, and then a policy change occurs. Is there any relation between the "old" value and the "new"? How much information can be reused in a re-computation of new the trust value?

Approximation The idea of approximation is related to issues robustness, complexity and dynamics. By an approximation of the least fixed point, we mean a value $m : \mathcal{P} \to \mathcal{P} \to T$ which is somehow related (e.g. by one of the orderings $\sqsubseteq, \preceq$) to the actual least fixed point $\mathsf{lfp}\ \Pi$. In some situations principals may not need to compute the entire fixed point – a sufficiently good approximation may be enough for a decision to be made. Applying approximation techniques may reduce the complexity and enhance robustness of a computation. Approximation is also related to the area of abstract interpretation.

Initiation *When* is a fixed point computation initiated? This may be *on demand*, e.g. when p needs to reason about its trust in q. Or it may be *continuously ongoing*, so that fixed point computations are run always, and values are available when needed. Also, *who* initiates computation? This may be the principal that wants to compute its value, but, in contrast, computation could also be initiated by other principals, e.g. as new information become available to them.

Process: Freeze phase for root node R
$R.S \leftarrow \emptyset$; $R^- \leftarrow \emptyset$;
$||_{\{\texttt{A},\texttt{B}\}}$
$\texttt{A}$: **replicate**
[**receive** ($\texttt{mark}$) **from** Y;
$R^- \leftarrow R^- \cup \{Y\}$;
send ($\texttt{ack}$, $\texttt{ok}$) **to** Y]

$\texttt{B}$: $||_{c \in R^+}$
c : **send** ($\texttt{mark}$) **to** c;
receive ($\texttt{ack}$, M) **from** c;
if (M=$\texttt{marker}$) **then** $R.S \leftarrow R.S \cup \{c\}$
|| **join** (R^+) **then** "end freeze phase"

Figure 6: Freeze algorithm - Root node behaviour

4.1 Distributed Algorithms

We use an asynchronous communication model. The nodes communicate by asynchronous message passing: we assume no known bound on the time it takes for a sent message to arrive. We do assume that communication is reliable in the sense that any message sent eventually arrives, exactly once, unchanged, to the right node, and that messages arrive in the order in which they are sent. We assume that all nodes are willing to participate in the computation, that they do not fail, and that they have unbounded memory. We discuss relaxations of these requirements later.

The algorithms all have a more or less common initialisation phase, called the *freeze phase*. The algorithms differ in what happens after this phase, that is, in how the actual computation proceeds.

Freeze phase

Consider the subgraph $G_R \leq G$ of nodes and edges reachable from R in the dependency graph G. Assume without loss of generality that this graph is $G_R = ([n], E)$, $n \leq m$, $E \subseteq D$, $E : [n] \to \mathbf{2}^{[n]}$. The freeze phase has two goals which are to be achieved at the end of the phase. Firstly, each node must obtain a list of the nodes that depend on it. Secondly, we want to compute a spanning tree $T_R \leq G_R$ with root R, so that each node knows its parent and its children in this tree. We denote $T_R = ([n], S)$, $S \subseteq D$, $S : [n] \to \mathbf{2}^{[n]}$.

For any node i, we denote the set of nodes j for which $j \in E(i)$ by i^+ and the set of nodes k for which $i \in E(k)$ (i.e. $E^{-1}(\{i\})$), by i^-. So to summarise, after the freeze phase, any node i knows i^+ and i^-, and it knows its parent p_i and its children $S(i)$ in a spanning tree T_R rooted at R. Node i will store i^+ and i^- in variables of the same name, and will store p_i and $S(i)$ in variables $i.p$ and $i.S$.

In the case of trust value computations, this phase can also ensure that the functions π_i are "frozen" for the rest of the computation, meaning that if the functions change e.g. due to a policy change or a local trust values change, then instead, the old function is used to ensure consistency of the computation.

The distributed algorithm for the freeze-phase is described by a process that runs at each node. The root node runs a special process given in Figure 6, and all other nodes run a similar process given by Figure 7. One way to understand the algorithm is to think of it as a distributed graph-marking algorithm: the initial message that a node i receives from a node j "marks" the node i, and j is then the "marker" for i. Then the edges between "marker" and "marked" nodes, will

Process: Freeze phase for node i
receive (`mark`) **from** X;
$i^- \leftarrow \{X\}$; $i.p \leftarrow X$; $i.S \leftarrow \emptyset$
$||_{\{\texttt{A},\texttt{B}\}}$
`A` : **replicate**
[**receive** (`mark`) **from** Y;
$i^- \leftarrow i^- \cup \{Y\}$;
send (`ack`, `ok`) **to** Y]

`B` : $||_{c \in i^+}$
c : **send** (`mark`) **to** c;
receive (`ack`, M) **from** c;
if (M=`marker`) **then** $i.S \leftarrow i.S \cup \{c\}$
|| **join** (i^+) **then send** (`ack`, `marker`) **to** x

Figure 7: Freeze algorithm - Generic node behaviour

constitute the spanning tree T_R. Furthermore, once a node is marked it starts a "server" sub-process (labelled `A`) which accepts `mark`-messages from any node Y, adds Y to its dependency set i^-, and acknowledges with an "`ok`" message. A sub-process running in parallel (`B`), notifies all nodes that i depends on (i.e. i^+) of this dependency, and waits for each node to acknowledge. This acknowledgement is either "`ok`" in case i is not the marker, or "`marker`" in case i *is* the marker. Finally, when an acknowledgement has been received from each child, i acknowledges the node that sent it the initial message with a `marker` message.

The following statements hold. The number of messages sent is $O(|E|)$, each message of bit length $O(1)$. This follows from the observation that for each edge in the graph there flows at most two messages, one `mark` and one acknowledgement. When the root node R has received acknowledgement from all its children then every node i which is reachable from R stores in the variable i^-, the set i^- (by abuse of notation), stores in variable $i.S$, the children of i in T_R, and in variable $i.p$, i's parent in T_R. Note that we only mark the nodes that are reachable from R, this means that hopefully we are cutting off a large piece of the nodes $[n]$, which will not need to be involved in the computation phase.

Globally synchronous computation

We describe here briefly an algorithm which is globally synchronised by the root node. We assume that each node x initially has the following data allocated: the sets x^-, x^+ along with the spanning tree data, $x.S$ and $x.p$, as resulting from the freeze-phase. We assume that x also allocates a T array, $x.m$, of size $|x^+|$ and a boolean array $x.b$ of size $|x.S|$. Initially, $x.m$ is $\bot_\sqsubseteq$ in all entries and $x.b$ is false in all entries. Node x also has variables $x.t_{cur}$ and $x.t_{old}$ of type T, each storing $\bot_\sqsubseteq$ initially.

The algorithm then proceeds by a number of rounds. Each round is initiated by the root node, R, which sends a round-initiation message to each of its children in the spanning tree, which in turn forward it to each *its* children, and so on "recursively". Once a node x has been initiated, it starts by saving the last estimate $x.t_{cur}$ in variable $x.t_{old}$. It then computes its function f_x, with respect to the values that are currently stored in the array m. This computed value is then stored in $x.t_{cur}$. If $x.t_{cur} \neq x.t_{old}$ then $x.t_{cur}$ is sent to all nodes that depend on it, i.e. each node $d \in x^-$, which stores this value in $d.m[x]$. Node x also expects to receive a boolean value from each node $c \in x.S$ which is stored in the array b, in entry c. The invariant will be that $b[c] = \mathsf{true}$ iff there was some change in the sub-tree rooted at c. Node x

then sends to its parent $x.p$ in the spanning tree, the value true iff $x.t_{cur} \neq x.t_{old}$ or some $b[c]$ is true. Once the root has received all the boolean values that it expects, the round is done. If the root node computed the value `false`, then it holds globally that there was no change, and the least fixed point has been reached. Otherwise the next round is initiated.

We can prove the following invariant. For any node x, and for any integer $i \in \mathbb{N}$, then after the i'th round, we have $x.t_{cur} = (F^i(\bot_{\sqsubseteq}))_x$, and for all $y \in x^+$, $x.m[y] = y.t_{cur}$. Here "after the 0'th round" means initially. Furthermore, for any $c \in x.S$, the value $x.b[c]$ is true after the i'th round if and only if there exist a node y in the subtree of T_R, rooted at child c, which has $y.t_{cur} \neq y.t_{old}$. This implies that if after a round $i > 0$, all entries in $R.b$ are false, and $R.t_{cur} = R.t_{old}$, then the value $R.t_{cur}$ stores a fixed point value for R. Furthermore, because of the finite height h of lattice T, this occurs after some iteration $i \leq n \cdot h$, and if i is the first round in which this occurs, then $R.t_{cur}$ stores the local least-fixed-point value for R. To see this, note that the invariant holds initially. Assume now that the invariant holds after round $i \geq 0$, prove that after round $i + 1$ this holds, and that each round always terminates. Since T_R forms a spanning tree, each node in G_R will eventually be initiated. By the invariant, for any node, x, once it receives the initiation message, it will compute its function f_x, on the array $x.m$. The invariant implies that the array stores the components x^+ of $F^i(\bot_{\sqsubseteq})$ (we assume that any "update" messages received prior to this evaluation is stored in a special array, and only transferred to m after f_x has been evaluated), clearly the computed value is $f_x(F^i(\bot)) = F^{i+1}(\bot)_x$. Since this is the value that is sent to all nodes x^-, this reestablishes the "m"-part of the invariant. The round terminates once the root has received the boolean values from each of its children. Because T_R is a spanning tree, the boolean values will flow upwards from the leaves of this tree, towards to root. Clearly this will terminate, and the desired property holds.

The algorithm can be described as a distributed analogue of the Jacobi-iterations $\bot, F(\bot), F^2(\bot), \ldots$. The algorithm is globally synchronous in the following sense: in each round all nodes wait for the round-initiation message before computing its function. This message is only received when all nodes in the graph reachable from R, has computed its function in the previous round, and has sent the result to all that depend on it. So in a sense, the computation is as slow as the slowest node in the graph. We see that for each edge in the dependency graph reachable from R, $G_R = ([n], E)$ there will flow at most h messages containing T-values. This is because any node sends a message only if its current value changes, and by monotonicity this can happen at most h times at each node before the least fixed point is reached. With respect to the boolean messages, in each round every node except for the root sends exactly one bit to its parent in the spanning tree. Since the number of rounds is bounded by the height of the lattice T^n, which is $h \cdot n$, the number of boolean messages is bounded by $h \cdot n^2$. The total the number of messages sent is $O(h \cdot (|E| + n^2)) = O(h \cdot n^2)$. The number of bits sent is $O(h \cdot |E| \cdot \log_2 |T| + h \cdot n^2)$. Each node x will locally compute its function $O(h \cdot |x^+|)$ times, since it need only be re-evaluated if one of the input values actually changes.

Totally asynchronous computation

We describe an algorithm which is essentially described and proved correct in Bertsekas' book [34]. In this algorithm, there is no notion of 'round', each node simply computes its function at its own pace, and does this *each time* a new value arrives, instead of waiting for an entire round to complete. Any nodes is always in one of two states, *sleep* or *wake*. If a node is in *sleep* state, then the reception of a message triggers a transition to the *wake* state. All nodes start by making a transition from the *sleep* state to the *wake* state. Each node x has the lists x^+ and x^-, and has a

T array of size x^-, which is initially $\bot_\sqsubseteq$. In the *wake* state any node x repeats the following: it starts by computing its function to the values in the T-array. If there was no change in the resulting value (compared to the last value computed), it will go to the *sleep* state unless a message was received since it was last sleeping. Otherwise if a new value resulted from the computation, this value is sent to all nodes in x^-. Concurrently with this, we run a termination detection algorithm, which will detect when all nodes are in the *sleep*-state, and no messages are in transit. Dijkstra's termination detection algorithm [39] does this, and Bertsekas has also an efficient algorithm [34].

This simple asynchronous behaviour gives a correct algorithm by the monotonicity assumptions. Furthermore, we can prove that since any node x sends values only when a change occurs, by monotonicity of the f_x function, x will send at most $h \cdot |x^-|$ messages, each of size $O(\log_2 |T|)$ bits. Node x will receive at most $h \cdot |x^+|$ messages, and in the worst case it will do as many computations of f_x. Globally, the number of messages sent is $O(h \cdot |E|)$ each of bit size $O(\log_2 |T|)$. The cost of the termination detection scheme must be added to this.

Ideas towards a minimal communication complexity algorithm for certain graphs

We present briefly an idea for an optimisation which can improve the message and computation complexity for some graphs. The idea is to compute the strongly connected components in the graph G_R. Suppose that we have an efficient distributed algorithm, which can do the following. After the algorithm terminates, each node x has a number $c_x \in \mathbb{N}$ and a boolean l_x. These satisfy the property that for any maximal strongly connected component (scc) C in G_R, then for all $x, y \in C$: $c_x = c_y$, and there is a unique leader $z \in C$ with $l_z = \mathsf{true}$. In other words, we have distributedly computed the sccs of the dependency graph G_R, each node knows which scc it belongs to, and each scc has a unique leader node $L(C)$. Now consider any topological ordering of the graph, where we collapse all sccs into just their leader nodes. The fixed point computation can then proceed by the topological ordering: each scc waits until all sccs that it depends on has computed its fixed point value. Once this is done it will compute its fixed point value and send this to the sccs that depend on it. This approach means that we are not wasting messages on sending intermediate results, instead, we send the result for a scc only once it is completely computed. Provided that the algorithm for scc-computation is not too expensive, this will have a better message complexity than any of the other algorithms for many non strongly-connected graphs.

4.2 Considerations

We have discussed algorithms for the distributed computation of order-based fixed points. The computation of least fixed points, parallelises very well, and the algorithms described are reasonably efficient. Our main motivation for studying the distributed computation of local least fixed points is that the semantics of the policies in our trust framework is given by least fixed points. It is clearly necessary for a principal to be able to compute or at least approximate, its trust in other principals. However, since fixed points are pervasive in computer science, there may easily be many other applications of such distributed algorithms. Our concern is trust based applications so we consider issues pertaining to this.

Robustness Suppose a node is not willing to participate in the computation. This is then discovered in the freeze-phase. One may replace all policy references to that node by the bottom element $\bot_\sqsubseteq$. If this is done, we will reach a fixed

point, m, which will be an approximation to the actual value, $m \sqsubseteq \mathsf{lfp}\ F$, that would have been computed if the node would participate. This holds for both the globally synchronous algorithm and the totally asynchronous algorithm. If instead a node fails *during* computation, this will a priori delay the globally synchronous algorithm indefinitely. The asynchronous algorithm is more robust: computation simply proceeds with the last value transmitted by the failing node. This ensures that one obtains a fixed point value which will be an approximation to the real fixed point value.

Complexity and scalability The complexity of the algorithm is not bad. The computation and communication load can, in part, be balanced by the number of principals listed as references in local policies, which is nice because it means that poor principals can locally adjust resource consumption by simplifying their policies. Another approach could be for poor principals to submit their trust policies to a trusted party to represent them in the computation, e.g. a computation server. There is one potential problem with the scalability of our approach, which is sometimes referred to as the "small world" or "six degrees of separation" phenomena: according to legend, every pair of human beings is connected by at most six "a friend of"-links. If the same phenomena occurs with trust policies, it does not help much to try and compute local fixed points. One approach might be to cut off computation at some fixed "reference-distance" to the root node.

Security In contrast to usual reputation systems, in which *every* principal contributes to the "reputation value" of every other principal, our approach is based on local trust policies. This means that in order for a group X of colluding principals to affect the trust value of a principal p for another principal x, they would need to know p's trust policy entry for x, and then "corrupt" principals that p depends on. This seems harder than in classical reputation systems. There is another, and it seems more severe, problem which is related to the so-called "Sybil"-attack [40, 41]. The problem occurs in distributed systems, that allow change of identity, or the acquisition of several identities[13]. If one assigns more privilege to unknown principals than to principals which are known, but have behaved "badly", then it is possible for principals to simply create a new identity, effectively erasing their history. Currently it is not clear what to do to prevent this, other than ensuring that there is a high cost associated with creating new identities. Identity thefts or purchases might also present problems.

Privacy It is not clear how to completely protect the privacy of principals. Even though principals can keep their policies secret, when they participate in a fixed point computation, the input-output behaviour of their policy can be queried, and by tracing messages, one may learn which principals the policy depends on. Cryptography and anonymity techniques like onion routing [42] may alleviate this.

Dynamics The freeze-algorithm ensures the consistency of the computation: if a policy changes during computation, one uses the old policy for the rest of the computation, instead of the new. Interesting future research is to formalise exactly what constitutes a policy update, which may allow application of the area of dynamic algorithms for least-fixed-point computation. As an example, if a policy update is information increasing, as is the case when observing an

[13] The name "Sybil" refers to a Sybil Dorsett, which was the first person to be diagnosed with multiple personality disorder. Apparently, Dorsett hosted 16 distinct personalities before she recovered.

event in the SECURE framework, then we conjecture that one may continue computation with the updated policy instead of the old.

Approximation As stated above, if one accepts approximations of the least fixed point, we can cope with failures and certain policy updates. There is also the option to stop computation when a "good enough" value has been reached or time has run out. Techniques like abstract interpretation may also be useful.

Initiation All our algorithms are initiated by the root node, that is, the node that wants to compute its trust value. Another approach is to have computation run constantly: nodes are continuously participating in an ongoing computation, e.g. by running the asynchronous algorithm without termination detection! If communication resources are abundant this may increase the speed of decision-making since trust values are simply always available.

5 Conclusion and research directions

We have presented a mathematical framework for trust, and shown that this has many useful instantiations. We have sketched a preliminary axiomatisation of a category of trust structures, and confirmed that many of our examples seem to naturally satisfy these axioms. We have stated without proof the result that the interval construction can be explained as the left adjoint in a coreflection of our category of trust structures in a category of complete lattices. The formal trust model in the SECURE project is an instance of our framework, and with this model we have found a novel use of event structures, traditionally studied in true concurrency theory. We have proposed a generalised notion of event-structure morphisms to achieve an expressive mechanism for transfer of information between related contexts. Finally we have presented and analysed distributed algorithms for local least-fixed-point computation, indicating a possible implementation for concrete dynamic trust-management systems.

It is interesting to note how wide the subject of dynamic trust management spans across the field of computer science. On the theoretical side we have used elements from the theory of partially ordered sets and complete lattices, and very importantly, the ideas on "information" from domain theory are applied to model uncertainty, and give denotational semantics to policy languages. Category theory provides the framework for understanding the interval construction, and a language for formalisation of the notion of trust structures in terms of a category of interest in itself. The SECURE model uses the notion of event structures from concurrency theory, and relies on the possibility of deriving (uncertain) probabilities on event-structure configurations, which is related to the notion of probabilistic event structures. While we are not invoking or inventing deep theorems, we have found nice new applications of the tool-belt of theoretical computer science, reinforcing the power of its theories and techniques. On the more practical side there are also many issues of interest. The fields of algorithms, and distributed and parallel computation are clearly relevant. The traditional field of computer security may be of interest to protect the privacy of involved principals, and perhaps the secrecy of their policies. Dynamic algorithms seem a promising area to increase efficiency of algorithms for trust-value computations. Approximation algorithms and the field of abstract interpretation may also be useful to this point. We have chosen to focus on algorithms needed for implementation of real dynamic trust-management systems, enabling the actual *computation* of trust values. While it is clear that not all issues have been resolved, it is at least reassuring to note how well algorithms for local least-fixed-point computations distribute. The work of Bertsekas [34] has been very helpful here, providing the basic algorithms which need only be combined with the

ideas on local fixed points, to obtain useful distributed algorithms. General distributed algorithms for local least-fixed-point computation may very well be useful in other areas, e.g. using distribution to alleviate the state explosion problem of model checking, and perhaps distributed static analysis of programs could be of interest.

5.1 Thoughts about the framework

The key concept in our mathematical framework is the notion of an *information ordering*, that may possibly be distinct from the trust ordering. For the interval construction, an interval $[d, e]$ is information-wise less than $[d', e']$ if $[d, e]$ can be narrowed into $[d', e']$, i.e. $d \leq d'$ and $e' \leq e$. This allows for the specification of *uncertain* trust values, e.g. $[d, e]$ means that the "real" trust value t is above d but below e. When one learns more about the behaviour of the principal in question, one may narrow the interval. In this case all "trust updates" are information-increasing. This is certainly a nice property, but for some intervals the natural "trust update" is not necessarily information-increasing. Consider again the Jøsang-example (2.4) of the framework from Section 2. Here trust values are intervals $[b, b+u] \subseteq [0, 1]$ of reals. One may think of such an opinion triple (b, u, d) as originating from (sic)-triples, e.g by a normalising mapping $(s, i, c) \mapsto b = \frac{s}{s+i+c+1}, u = \frac{i+1}{s+i+c+1}, d = \frac{c}{s+i+c+1}$ - this gives an interval in which the uncertainty decreases as $s + c$ increases, and the lower bound increases with s and the upper bound increases with c. However, in this case, one natural type of trust update is to map $s \mapsto s+1$, $c \mapsto c+1$ or possibly $i \mapsto i+1$, which results in a new interval which is unrelated to the old interval with respect to the usual reverse subset-inclusion information order. This mismatch can be explained by making a distinction between two sources of uncertainty, and two corresponding types of trust-updates. The first type of update is something which *refines* information which is already, in some sense, present. In the SECURE framework, an example of this is to add an event e to a non-maximal configuration x_i in an interaction history. In terms of (sic)-triples, this *distributes* values from the i-component to s or c components. The other type of update occurs instead when a new interaction is recorded. This is the type of "non-increasing" update, mentioned above, which adds 1 to the triple. The information ordering on intervals fits perfectly with the first type of update, the refinement, but less so with the second type. One might use the terms *quality of information* for the internal uncertainty (the i-component for (sic)-triples) and the term *quantity of information* for the other. These two sources of uncertainty, quality and quantity of information, seem to be orthogonal, e.g. one may increase the quality without affecting the quantity or vice versa. It will be interesting future research to see if it makes sense to include this notion of quantity as explicitly modelled in trust structures. One approach might be to add to a structure $(X, X_!, \sqsubseteq, \preceq)$ a *measure*, $\mu : X \to [0, \infty]$, which assigns the quantity of information to an element. It is interesting to note that in many examples the *dogmatic* points d then have the property that they are $\sqsubseteq$-maximal within the class of elements that have measure $\mu(d)$ or less. A subject of further investigation.

5.2 Future work?

The work presented in this report is still at an early stage. Most of the work presented is the result of purely theoretical considerations. Fortunately, the participation of University of Aarhus in the SECURE project has complemented the theoretical work with also practical validation: an actual Java implementation of the SECURE framework, which is based on the trust model from Section 3 will be available as an Open Source project in the near future. This includes also some

example applications of the framework, e.g. spam filtering [16] and secure PDA collaboration [43].

One may very reasonably ask what this mathematical formalisation and theory can really provide the computer science community interested in trust-based computer security or decision-making. First of all it can bring order into chaos! The amount of "trust-related" work published in the computer science community (typically practically oriented security- or P2P- and distributed-systems-communities) is enormous. Many papers follow the same pattern: a short introduction explaining why "trust" is important, a brief outlining of some other systems, and most of the space is used to present some *new* system which usually has some distinctive characteristic. A good formal framework could allow for uniform comparison of systems, and enhance the understanding of what trust in computer systems *essentially* is. Secondly, and probably more importantly, with a formalisation of the essential, characteristic properties of such trust based systems, comes the ability to formalise and prove properties of these. Interesting properties could be "security" properties which have the flavour: "if trust policy π assigns trust value t to principal q then q has a past behaviour b which satisfies property $\phi(b)$". Or dually, "liveness" or "availability" properties like: "if principal q has past behaviour b which satisfies property $\phi(b)$, then trust policy π always assigns to q, at least value t". To be able to prove such properties, one must come up with proper, universally acceptable formalisations of concepts like "trust policy", "trust value" and "behaviour".

One might compare the state of current trust-based research to the history of other areas of computer science. Consider for example the area of cryptography. In early crypto research, one may see researchers present crypto-schemes (e.g. DES) and prove nice properties, e.g. encryption E and decryption D satisfy $D \circ E = Id$, and perhaps supply heuristic analyses which seem to indicate that encrypted messages are well-hidden. But what does it really mean for a crypto-system to be *secure*? With time the theoretical community seems to have agreed on "good" definitions, e.g. many security definitions are based on some form of indistinguishability of ideal functionality from the actual system. This allows one to propose a crypto system and then *prove it secure* relative to a definition (and usually some hardness assumption!). Similarly, one might ask the question: what does it mean for a dynamic trust-based system to be *secure* or *correct*? (or whatever the right term is.)

Formal methods may also provide the basis for proofs of correctness of small protocols which can be used, e.g. for optimisation. To illustrate what we have in mind here, we consider a small example of such a protocol. Consider the following result.

An approximation result

Let $T = (X, X_!, \preceq, \sqsubseteq)$, be a trust structure, and $C = \{x_i \mid i \in \mathbb{N}\}$ denote a countable $\sqsubseteq$-chain. If it is the case that for any $x \in X$ with $x \preceq C$, then also $x \preceq \bigsqcup C$, then $\preceq$ can be said to be C-$\sqsubseteq$-continuous. Say that $\preceq$ is $\sqsubseteq$-continuous if for any countable $\sqsubseteq$-chain C we have that $\preceq$ is C-$\sqsubseteq$-continuous.

Proposition 5.1. *Let $(X, X_!, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $t \in X$ and $\Pi : X \to X$ any function that is $\sqsubseteq$-continuous and $\preceq$-monotone. If the following conditions hold:*

- $t \preceq \bot_{\sqsubseteq}$
- $t \preceq \Pi(t)$

then

- $t \preceq \mathtt{lfp}_{\sqsubseteq}\Pi$.

Proof. We have $t \preceq \bot_{\sqsubseteq}$ which implies $\Pi(t) \preceq \Pi(\bot_{\sqsubseteq})$ by $\preceq$-monotonicity. Since $t \preceq \Pi(t)$, transitivity implies that $t \preceq \Pi(t) \preceq \Pi(\bot_{\sqsubseteq})$. So again by $\preceq$-monotonicity of Π and transitivity

$$t \preceq \Pi(t) \preceq \Pi^2(t) \preceq \Pi^2(\bot_{\sqsubseteq})$$

as a diagram:

$$\begin{array}{ccccccccc}
t & \xrightarrow{\preceq} & \Pi(t) & \xrightarrow{\preceq} & \Pi^2(t) & \xrightarrow{\preceq} & \cdots & \xrightarrow{\preceq} & \Pi^i(t) \\
\downarrow{\scriptstyle \preceq} & & \downarrow{\scriptstyle \preceq} & & \downarrow{\scriptstyle \preceq} & & & & \downarrow{\scriptstyle \preceq} \\
\bot_{\sqsubseteq} & \xrightarrow{\sqsubseteq} & \Pi(\bot_{\sqsubseteq}) & \xrightarrow{\sqsubseteq} & \Pi^2(\bot_{\sqsubseteq}) & \xrightarrow{\sqsubseteq} & \cdots & \xrightarrow{\sqsubseteq} & \Pi^i(\bot_{\sqsubseteq})
\end{array}$$

Now since for all $i \geq 0$ we have $t \preceq \Pi^i(\bot_{\sqsubseteq})$, the fact that $\preceq$ is $\sqsubseteq$-continuous implies that

$$t \preceq \bigsqcup_i \Pi^i(\bot_{\sqsubseteq}) = \mathtt{lfp}_{\sqsubseteq}\Pi$$

□

This proposition could be the basis of an efficient protocol for a form of "proof-carrying authorisation". Consider for simplicity the mn-trust structure (which satisfies that $\preceq_{T_{MN}}$ is $\sqsubseteq_{T_{MN}}$-continuous), and assume that we have a collection of policies Π which satisfies the requirements of our proposition. Suppose principal p wants to efficiently convince principal v that v's trust value for p is a pair, (m, n), with the property that n is less than some $N \in \mathbb{N}$, that is, it gives an upper bound on the amount of "bad" behaviour of p. Let us assume that v's trust policy π_v depends on a large set S of other principals. Assume also that it is sufficient that principals a and b in S have a reasonably "good" value for p, to ensure that v's value for p is not too "bad". If p knows that it has previously performed well with a and b, and knows also that v depends on a and b, it can engage in the following protocol: p sends to v the trust values $t = [p \mapsto (0, N), a \mapsto (0, N_a), b \mapsto (0, N_b)]$. Upon reception, v first extends t to a global trust state $\bar{t}$, which is the extension of t to a function of type $\mathcal{P} \to \mathcal{P} \to T_{MN}$, given by

$$\bar{t} = \lambda x \in \mathcal{P} \lambda y \in \mathcal{P}. \begin{cases} (0, N) & \text{if } x = v \text{ and } y = p \\ (0, N_a) & \text{if } x = a \text{ and } y = p \\ (0, N_b) & \text{if } x = b \text{ and } y = p \\ \bot_{\preceq} & \text{otherwise} \end{cases}$$

Principal p wants to verify that $\bar{t}(x)(y) \preceq (0, 0) = \bot_{\sqsubseteq}$ for all x, y: this is trivial if $y \neq p$ or $x \neq v, a, b$ because then $\bar{t}(x)(y) = \bot_{\preceq}$. For the other few entries it is simply an order theoretic comparison $\bar{t}(x)(y) \preceq (0, 0)$. Now v tries to verify that $\bar{t} \preceq \Pi(\bar{t})$. To do this, v verifies that $(0, N) \preceq \pi_v(\bar{t})(p)$. If this holds then it sends the value t to a and b, and ask a and b to perform a similar verification, e.g $(0, N_a) \preceq \pi_a(\bar{t})(p)$. Then a and b reply with *yes* if this holds and *no* otherwise. If both of a and b reply *yes*, then p is sure that $\bar{t} \preceq \Pi(\bar{t})$: by the checks made by v, a and b, we have that $\bar{t}(x)(y) \preceq \Pi(\bar{t})(x)(y)$ holds for pairs $(x, y) = (v, p), (a, p), (b, p)$, but for all other values it holds trivially since $\bar{t}$ is the $\preceq$-bottom on these. So by Proposition 5.1, we have $\bar{t} \preceq \mathsf{lfp}\ \Pi$, and so p is *ensured* that its trust value for v is above $(0, N)$.

The protocol has very much the flavour of a proof-carrying authorisation: the requester (or prover) must provide a proof that its request should be granted – it is then the job of the service-provider (or verifier) to check that the proof is correct. The strength of this protocol lies in replacing an entire fixed-point computation with

a few local check made by the verifier, together with a few checks made by a subset of the principals that the verifier depends on. A nice property of this protocol is that the information which the prover needs to supply should be available locally to the prover – it should already know who it has performed well with in the past. An important restriction of this approach is that because of the requirement that $t \preceq \bot_{\sqsubseteq}$, it can usually only be used to prove properties stating "not too much bad behaviour", and not properties guaranteeing sufficient good behaviour. It would be interesting research to see if other types of protocols allowing this can be found.

Acknowledgements

Author is supported by the SECURE project, EU FET-GC IST-2001-32486, and would like to thank everyone in the consortium for their cooperation, in particular consortium partners Andrew Twigg from University of Cambridge for helpful discussions about fixed point algorithms, and Marco Carbone, University of Aarhus. I would like to thank very much my excellent supervisor, Mogens Nielsen for his contributions and endless ideas and patience. I thank also Vladimiro Sassone for helpful discussions and the hint to look at probabilistic event structures.

References

[1] Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The role of trust management in distributed systems security. In Vitek, J., Jensen, C.D., eds.: Secure Internet Programming: Issues in Distributed and Mobile Object Systems. Volume 1603 of Lecture Notes in Computer Science., Springer (1999) 185–210

[2] Grandison, T., Sloman, M.: A survey of trust in internet applications. IEEE Communications Surveys & Tutorials **3** (2000)

[3] Blaze, M., Ioannidis, J., Keromytis, A.D.: Offline micropayments without trusted hardware. In Syverson, P.F., ed.: Financial Cryptography, 5th International Conference, FC 2001, Grand Cayman, British West Indies, February 19-22, 2002, Proceedings. Volume 2339 of Lecture Notes in Computer Science., Springer (2002)

[4] Blaze, M., Ioannidis, J., Keromytis, A.D.: Trust management for ipsec. In: Network and Distributed System Security Symposium (NDSS'01) Conference Proceedings. (2001) `http://www.isoc.org/isoc/conferences/ndss/01/`.

[5] Carbone, M., Nielsen, M., Sassone, V.: A formal model for trust in dynamic networks. In: Proceedings from Software Engineering and Formal Methods, SEFM'03, IEEE Computer Society Press (2003)

[6] Nielsen, M., Krukow, K.: Towards a formal notion of trust. In: Proceedings of the 5th ACM SIGPLAN international conference on Principles and Practice of Declarative Programming, ACM Press (2003) 4–7

[7] Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proceedings of the 17th Symposium on Security and Privacy, Oakland, California, IEEE Computer Society Press (1996) 164–173

[8] Blaze, M., Feigenbaum, J., Strauss, M.: Compliance checking in the policymaker trust management system. In: Financial Cryptography: Second International Conference, FC'98, Anguilla, British West Indies, February 1998. Proceedings. (1998) 254–274

[9] Weeks, S.: Understanding trust management systems. In: Proceedings from the IEEE Symposium on Security and Privacy, Oakland, California. (2001) 94–106

[10] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomsa, B., Ylonen, T.: SPKI Certificate Theory. RFC 2693, ftp-site: `ftp://ftp.rfc-editor.org/in-notes/rfc2693.txt` (1999)

[11] Blaze, M., Feigenbaum, J., Keromyti, A.D.: KeyNote: Trust management for public-key infrastructures. In: Security Protocols: 6th International Workshop, Cambridge, UK, April 1998. Proceedings. Volume 1550. (1999) 59–63

[12] Blaze, M., Feigenbaum, J., Keromyti, A.D.: The KeyNote trust management system, version 2. RFC-2704, ftp-site: `ftp://ftp.rfc-editor.org/in-notes/rfc2704.txt` (1999)

[13] Mac Lane, S.: Categories for the Working Mathematician. Graduate Texts in Mathematics. Springer-Verlag New York Inc. (1971)

[14] Winskel, G., Nielsen, M.: Models for concurrency. In Abramsky, S., Gabbay, D.M., Maibaum, T.S.E., eds.: Handbook of Logic in Computer Science. Volume 4. Oxford University Press (1995) 1–148

[15] Nielsen, M., Krukow, K.: On the formal modelling of trust in reputation-based systems. To be published in Springer Lecture Notes in Computer Science. Available online: `http://www.brics.dk/~krukow/` (2004)

[16] Cahill, V., Signeur, J.M.: Secure environments for collaboration among ubiquitous roaming entities. Website: `http://secure.dsg.cs.tcd.ie` (2004)

[17] Cahill, V., et al.: Using trust for secure collaboration in uncertain environments. IEEE Pervasive Computing **2** (2003) 52–61

[18] Chu, Y.H., Feigenbaum, J., LaMacchia, B., Resnick, P., Strauss, M.: REFEREE: Trust management for (web) applications. Computer Networks and ISDN Systems **29** (1997) 953–964

[19] Jim, T.: SD3: A trust management system with certified evaluation. In: Proceedings from the IEEE Symposium on Security and Privacy, Oakland, California. (2001) 106–116

[20] Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. Theoretical Computer Science **13** (1981) 85–108

[21] Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press (1976)

[22] Lucas, van der Gaag: Principles of Expert Systems. Addison-Wesley (1991)

[23] Jøsang, A.: A logic for uncertain probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **9** (2001) 279–311

[24] Kulisch, U.W., Miranker, W.L.: Computer Arithmetic in Theory and Practice. Computer Science and Applied Mathematics. Academic Press Inc., New York (1981)

[25] Scott, D.S.: Domains for denotational semantics. ICALP '82 - LNCS **140** (1982)

[26] Varacca, D.: Two Denotational Models for Probabilistic Computation. PhD thesis, BRICS, Department of Computer Science, University of Aarhus, Denmark (2003)

[27] Varacca, D., Völzer, H., Winskel, G.: Probabilistic event structures and domains. To be published. Website: `http://www.brics.dk/~varacca/` (2003)

[28] Shmatikov, V., Talcott, C.: Reputation-based trust management. Accepted to: Journal of Computer Security (selected papers of WITS'03) (2004)

[29] Mui, L., Mohtashemi, M.: Notions of reputation in multi-agent systems: A review. In: Trust, Reputation, and Security: Theories and Practice, AAMAS 2002 International Workshop, Bologna, Italy, July 15, 2002, Selected and Invited Papers. (2002)

[30] Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Commun. ACM **43** (2000) 45–48

[31] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation managments in P2P networks. In: Proceedings of the twelft international conference on World Wide Web, Budapest, Hungary. (2003) 640–651

[32] Advogato Website: Advogato's trust metric. `http://www.advogato.org` (2000)

[33] Jøsang, A., Ismail, R.: The beta reputation system. In: Proceedings of the 15th Bled Conference on Electronic Commerce, Bled. (2002)

[34] Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice-Hall International Editions. Prentice-Hall, Inc. (1989)

[35] Winskel, G.: Formal Semantics of Programming Languages : an introduction. Foundations of computing. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts (1993)

[36] Grätzer, G.: Lattice Theory. W.H. Freeman, and Company (1971)

[37] PGPi website: An introduction to cryptography, in pgp user's guide 7.0. ftp: `ftp://ftp.pgpi.org/pub/pgp/7.0/docs/english/IntroToCrypto.pdf` website: `http://www.pgpi.org/doc` (2000)

[38] Vergauwen, B., Wauman, J., Lewi, J.: Efficient fixpoint computation. In Le Charlier, B., ed.: Static Analysis (SAS'94). Springer, Berlin, Heidelberg (1994) 314–328

[39] Dijkstra, E.W., Feijen, W.H.J., van Gasteren, A.J.M.: Derivation of a termination detection algorithm for distributed computations. Information Processing Letters **16** (1983) 217–219

[40] Douceur, J.R.: The sybil attack. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer-Verlag (2002) 251–260

[41] Friedman, E., Resnick, P.: The social cost of cheap pseudonyms. Journal of Economics and Managment Strategy **10** (2001) 173–199

[42] Syverson, P., Goldschlag, D., Reed, M.: Anonymous connections and onion routing. In: IEEE Symposion on Security and Privacy, Oakland, California. (1997) 44–54

[43] Shand, B., Dimmock, N., Bacon, J.: Trust for transparent, ubiquitous collaboration. In: First IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2003), Dallas-Ft. Worth, TX, USA (2003) 153–160