Shinren: Non-monotonic Trust Management for Distributed Systems

Changyu Dong, Naranker Dulay

Department of Computing
Imperial College London
180 Queen's Gate, London, SW7 2AZ, UK
{changyu.dong,n.dulay}@imperial.ac.uk

Abstract. The open and dynamic nature of modern distributed systems and pervasive environments presents significant challenges to security management. One solution may be trust management which utilises the notion of trust in order to specify and interpret security policies and make decisions on security-related actions. Most trust management systems assume monotonicity where additional information can only result in the increasing of trust. The monotonic assumption oversimplifies the real world by not considering negative information, thus it cannot handle many real world scenarios. In this paper we present Shinren[1], a novel non-monotonic trust management system based on bilattice theory and the any-world assumption. Shinren takes into account negative information and supports reasoning with incomplete information, uncertainty and inconsistency. Information from multiple sources such as credentials, recommendations, reputation and local knowledge can be used and combined in order to establish trust. Shinren also supports prioritisation which is important in decision making and resolving modality conflicts that are caused by non-monotonicity.

1 Introduction

The advances in communications and computing research have brought distributed systems and pervasive environments to new prominence. Applications are now distributed across the boundaries of networks, organisations, even countries and deployed on smaller mobile devices. The increasing scope of distributed applications also implies that applications must deal with "strangers" from other organisations and places. This leads to new challenges. How does the security system determine whether or not a request should be allowed if the request comes from an unknown user? The system must be able to decide without pre-knowledge of the user in order to authorise/deny the access. In other words, the system must determine whether and by how much does it trust a user. Trust management [1] was introduced in response to the challenges posed by modern distributed systems and pervasive enviroments.

In real-life, trust is normally **non-monotonic**. Consider the following:

"You are the CEO of a bank and looking for someone to manage a multi-billion pounds investment fund. A CV arrives on your computer. You quickly read through it:

[1] Shinren: the pronunciation of trust in Chinese

*worked for the UK's oldest investment bank (interesting), had more than ten years experience as a derivatives trader (**good**), was the Chief Trader and General Manager of operations in futures markets on the Singapore Monetary Exchange (**great**), made £10 million a year which accounted for 10% of former employer's annual income (**excellent**). You almost make up your mind. Then you see the candidate's name: Nick Leeson*[2]. *Everything is turned upside down. You trash the email."*

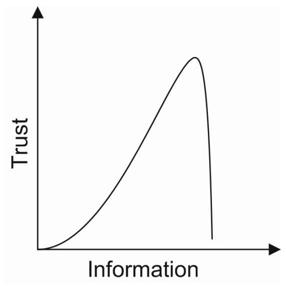

Fig. 1. Non-monotonic trust for CV

If we draw a diagram of this trust-information relation, it might look like Figure 1. From the diagram we can see that when new information comes in, trust can decrease, as well as increase, sometimes drastically. In other words, trust is non-monotonic. The non-monotonicity of trust is a natural consequence of the existence of both goodness and badness in the world. Trust, as defined by Mayer and Davis [2], is "the willingness of a party to be vulnerable to the actions of another party". To trust someone, the trustor needs to judge how competent, how honest, how dependable the trustee is, but more importantly, how incompetent, how dishonest and how undependable the trustee is. Positive information tells us how much we may gain from a trust relationship, while negative information tells us how much we may lose from it. Ignoring negative information may result in misplaced trust which may in turn cause serious damage to the trustor.

Although trust is non-monotonic, mainstream trust management systems [1, 3–7] are monotonic. The reason is that complete knowledge is hard to achieve in large distributed systems and also that those systems are based on classical logic which cannot cope with this situation. Classical logic is monotonic which means a conclusion will never be retracted with new information, i.e. if $\Gamma \models \phi$ then $\Gamma \cup \gamma \models \phi$. To model trust with classical logic, a monotonic assumption is introduced to solve the problem and simplify the design by not using negative information about the world. Monotonic systems do not have problems with incomplete information because all missing information is positive and every decision that they make can only be more "correct" with more information. Accepting the monotonic assumption means accepting the world is always positive (however if there were no negative things in the world, do we still need

[2] Nicholas Leeson, the infamous rogue trader who caused the collapse of Barings Bank.

trust management systems?). Monotonic trust management systems have many advantages, but the monotonic assumption is too limiting for many scenarios.

The gap between real world requirements and the current design of trust management systems motivates our work. A more realistic solution is needed for managing trust in distributed systems. In this paper, we describe Shinren, a novel non-monotonic trust management system based on bilattice theory and the any-world assumption. Shinren can make reasonable decisions even with incomplete information. Moreover, it can also utilise unreliable information which makes it more suitable for open distributed systems where reliable information is often hard to come by. Shinren does not just simply make use of the existing theories, it also supports prioritisation, which is achieved by a non-trivial extension of the original theories. Prioritisation is important in resolving conflicts and providing support for decision making.

This paper is organised as follows: We review related work in section 2 and compare Shinren with current non-monotonic trust management systems in section 3. We discuss the motivation of this work in section 4. In section 5 we introduce bilattice theory and the any-world assumption. In section 6, we describe Shinren, its policy language and semantics. In section 7 and 8, we present two examples to show the details of policy evaluation in Shinren. In section 9 we show a prototype implementation. Section 10 concludes the paper.

2 Related Work

Trust management has attracted a lot of interest from the research community and many trust management systems have been proposed. For example, PolicyMaker [1] and its successor KeyNote [8], Simple Distributed Security Infrastructure (SDSI) [9], Simple public key infrastructure (SPKI) [10], Query Certificate Manager (QCM) [11], the RT framework [5] and Cassandra [12]. All the above systems are monotonic. It has been shown in [8] that the correctness of PolicyMaker and Keynote is guaranteed only if all the policies are monotonic. In [13], the authors show how SDSI and SPKI rules can be represented in a monotonic logic and its extension. QCM is based on relational algebra and can be transformed into non-recursive Datalog. RT is based on Datalog. In [14], Li and Mitchell proposed an expressive extension of RT based on Datalog with constraints as the foundation for trust management languages. Datalog with constraints is also the semantic foundation of Cassandra. There are also several non-monotonic trust management systems which support particular negative policies: REFEREE [15], the Trust Establishment System from IBM [16] and $RT_{\ominus}$ [17]. We discuss and compare the existing non-monotonic trust management systems and Shinren in Section 3.

Many systems attempt to assign real values to trust and develop sophisticated mathematical models to calculate trust values [18–20]. The values are usually based on past experience. Although they are also called trust management systems, we view them as a totally different approach from the trust management systems presented above which rely mostly on logical reasoning and view trust decisions as logical consequences of certain facts and theories. Quantitative trust management systems are usually non-monotonic and can provide valuable information. However, the accuracy of the trust values largely depends on the amount of data input and may take a long time to get enough data. To differentiate, we call them reputation systems and Shinren can include such systems as subsystems.

Classical logic is the logical system which has been most intensively studied and most widely used. However, classical logical systems can only reason monotonically. The increase in knowledge can never invalidate the conclusions derived from what we originally know. This is fine in an well-defined mathematical system, but encounters problems in real world. The arguments on the limitations of classical logic can be traced back to Aristotle. There have been many formal non-monotonic logics and reasoning systems. For example, the Closed world Assumption (CWA) and its variants [21], Circumscription [22], Default Logic [23], Kleene's three-valued logic [24], Courteous Logic Programs (CLP) [25] and so on. Ginsberg's bilattice theory [26] provides a unified framework which can capture various forms of non-monotonic reasoning, e.g. default logic and assumption truth maintenance system.

Logic programming is another closely related topic. Many trust policy languages are based on or can be transformed into logic programs, and it has be widely accepted that real classical negation is infeasible for logic programming [27]. Usually only "negation as failure", a weaker form of negation, is implemented. Negation as failure, an approximation of the CWA, is by nature non-monotonic. However it does not make non-monotonic a favourable choice for trust management. One reason is that non-monotonicity poses difficulties in finding a standard declarative semantics for logic programming with negation. Several semantics have been proposed, including Clark's completion [28], Kripke-Kleene semantics [29], stable model semantics [30] and well-founded semantics [31]. It is hard to say which one is better than the others because they all have both advantages and limitations. Another problem is that these semantics, when used in building trust management systems, can take much more information from CWA than is desired and thus some unwanted conclusions may be derivable, as we will show in Section 3.

3 Comparison of Non-monotonic Trust Management Systems

Shinren is not the first non-monotonic trust management system. Rule-Controlled Environment For Evaluation of Rules and Everything Else (REFEREE) [15], the Trust Establishment System (TES) from IBM [16] and RT$_\ominus$ [17] are also non-monotonic. In Table 1, we summarise Shinren with these three main non-monotonic trust management systems. We also have some remarks on the table.

The main problem of existing non-monotonic trust management systems is semantics. A well-defined formal semantics is a critical part of any policy language. However, as we can see from Table 1, REFEREE and TES do not have formally defined semantics. RT$_\ominus$ is based on the well-founded semantics which is a non-monotonic semantics proposed originally for logic programming with negation [31]. The problem with using well-founded semantics in trust management is that it is based on CWA and the uniformity of CWA may lead to counter-intuitive results. For example, here is a simple trust policy $trust(a) :- \neg bad(a)$. Under the well-founded semantics, when $bad(a)$ is missing or not provable, it is falsified and thus makes $trust(a)$ true. However, this decision may seem too casual, especially when it is related to security. In Shinren, policy makers

	Shinren	**REFEREE**	**TES**	**RT$_\ominus$**
Truth Space	Bilattice	3-valued	2-valued	3-valued
Semantics	Extended Kripke-Kleene semantics over bilattice	Not Specified	Not Specified	Well-founded Semantics
Credentials	$\surd$	$\surd$	$\surd$	$\surd$
Recommendations	$\surd$	$\surd$	$\surd$	
Reputation	$\surd$	$\surd$		
Local knowledge	$\surd$	$\surd$		
Incomplete Information	AWA	OWA/CWA	CWA	CWA
Unreliable Information	$\surd$	$\surd$		
Contradictory Information	$\surd$			
Prioritisation	$\surd$			
Explicit Distrust Policies	$\surd$			

Table 1. Comparison of Non-monotonic Trust Management Systems

can use $unknown$ as the default value for $bad(a)$ while still use $false$ as the default value for other positive atoms.

Existing non-monotonic trust management systems are also less expressive than Shinren. For example, RT$_\ominus$ can only express policies using credentials. Among them, REFEREE is the most expressive one, however it has its own problem. It is capable of expressing policies utilising evidence from different sources, but it is incapable of distinguish decisions based on information of different quality. REFEREE is based on 3-valued logic, therefore there is no difference between a decision based on a statement from an authority and a decision based on rumour. The users of REFEREE may be given an unreliable trust decision without warning. In Shinren, a trust decision comes with a value which tells the user not only how true the decision is, but also how reliable it is.

4 Why Non-monotonic? Why Shinren?

So why do we need non-monotonic trust management systems? This is because (1) in the real world trust is non-monotonic and therefore a trust management system should be able to capture this; (2) monotonic assumption is not necessary in trust management, it is introduced merely because systems reasoning with classical logic cannot cope with the non-monotonicity in trust. The assumption does not solve the problem, it just makes systems ignore the problem. There are at least two bad consequences of the monotonic assumption: first, a trust management system which can be proved correct under the monotonic assumption may not be correct in the real world because the assumption does not hold in general; second, it makes trust management systems incapable of handling certain real world scenarios.

Under the monotonic assumption, monotonic trust management systems do not consider negative information. Syntactically, this is achieved by not allowing negations in the policies. Negation-free policies work fine in some cases, however they reflect a limited view of the world and are inappropriate in many cases. For example, negation-free policies are quite inconvenient in handling *exceptions*. In the world modelled by negation-free policies, it is quite hard to express, "trust all the police officers except the bad ones" because without negations, we would be allowed to say "trust police officers" but not "**do not** trust the bad police officers". In the extreme case, we must specify for each individual good police officer a trust policy in order to exclude the bad ones. Lacking the ability for specifying exceptions can be dangerous particularly in trust management systems where delegations are used. No exceptions means that decisions have to be fully delegated to a delegatee, and the system must fully accept the delegatee's opinions. No exceptions also means that the system cannot accept part of the delegatee's decision while declining other parts. In other words, the system loses control after delegation. Another case is that negation-free policies cannot handle *mutual exclusion*. Coke is tasty, orange juice is tasty too. But the mixture of the two does not taste so pleasant. There are many examples that are mutually exclusive. However, with negation-free policies, there is no way to express "A is good, B is good, but A+B is **not** good". In terms of security policies, separation of duties and conflict of interests are the most significant examples of this type of policy.

One may argue that in the real world, people try to hide their negative aspects. Therefore, even if policies are allowed to use negative information, if the system cannot find it, the non-monotonic feature is useless. It is true that the information we can collect is always limited. But consider the following:

$$\begin{array}{rl} \text{In monotonic trust management systems:} & trust\ :- good \\ \text{In Shinren:} & distrust\ :- bad \\ & trust\ :- good \end{array}$$

What is the difference? When the system cannot find bad, i.e. the negative information, Shinren can behave exactly as the monotonic ones. However, because it is not possible to use negative information in monotonic trust management systems, their decisions will still be trust even if bad is presented! In contrast, Shinren's decision will no longer be trust because the distrust policy is applied. Although not guaranteed, Shinren aims to limit any damage with its best effort approach rather than silently ignoring it.

By using bilattices, Shinren suffers less from a dilemma which all trust management systems must face: on the one hand, in order to make a correct trust decision, a large amount of information is needed; on the other hand, in order to make the decision correct, most of the information available cannot be used because it is not reliable. Shinren can reason with unreliable information even with contradictory information. Monotonic trust management systems cannot. This ability is especially important in acquiring negative information.

Prioritisation is not seen in any existing trust management system. The philosophy is that sometimes trust is not just a Yes/No decision, but also a choice. You might want to follow one rule even if there are multiple rules you can follow, you might trust someone even if there are several persons you can trust. Prioritisation allows policy makers to specify their preferences and thus make complex policies possible. And also, in the presence of modality conflicts, prioritisation seems to be the only way to resolve them. Although there are overheads in defining and managing policies when using prioritisation, the overheads are minimised in Shiren because policy makers only assign priorities to local policies and only trust (distrust) policies are prioritised.

5 Preliminaries

5.1 Bilattices

Bilattice theory [26] was introduced by Ginsberg in the 1980s, and has been widely used in non-monotonic reasoning, knowledge representation and artificial intelligence. Bilattice is a non-empty, possibly infinite set of values with two partial orders, each one giving the set the structure of a lattice. A lattice $\langle L, \preceq \rangle$ is a non empty set L along with a partial order $\preceq$ where any pair of elements $l_1, l_2 \in L$ has a least upper bound (join) and a greatest lower bound (meet) in terms of $\preceq$. We write $l_1 \prec l_2$ if $l_1 \preceq l_2$ and $l_1 \neq l_2$.

A bilattice, denoted by $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ where $\mathcal{B}$ is a non-empty set and $\preceq_t, \preceq_k$ are two partial orders called the truth-order and the knowledge-order respectively. $\preceq_t$ is an ordering on the "degree of truth". $b_1 \preceq_t b_2$ means b_2 represents at least as much truth as b_1 (and possibly more). Meet and join under $\preceq_t$ are denoted by $\wedge$ and $\vee$ and correspond to classical conjunction and disjunction. $\preceq_k$ is an ordering on the "degree of knowledge". Meet and join under $\preceq_k$ are denoted by $\otimes$ and $\oplus$. $b_1 \otimes b_2$ corresponds to the maximal information b_1 and b_2 can agree on, while $b_1 \oplus b_2$ combines the information represented by b_1 and b_2.

The class of bilattice we consider in this paper is restricted to *interlaced* bilattices. Interlaced bilattices are bilattices which satisfy the following: (1) if $b_1 \preceq_t b_2$ then $b_1 \otimes b_3 \preceq_t b_2 \otimes b_3$ and $b_1 \oplus b_3 \preceq_t b_2 \oplus b_3$; (2) if $b_1 \preceq_k b_2$ then $b_1 \wedge b_3 \preceq_k b_2 \wedge b_3$ and $b_1 \vee b_3 \preceq_k b_2 \vee b_3$. Thus in an interlaced bilattice an operation associated with one of the lattice orderings is required to be monotonic with respect to the other lattice ordering. This relates the two orderings. An alternative way of connecting the two orderings is via negation which reverses the truth ordering and is monotonic regarding the knowledge ordering.

Such bilattices can be constructed in a natural way by combining two lattices. Given two lattices $\langle L_1, \preceq_1 \rangle$ and $\langle L_2, \preceq_2 \rangle$, we can construct an interlaced bilattice as $\langle L_1 \times L_2, \preceq_t, \preceq_k \rangle$, where $(x_1, y_1) \preceq_t (x_2, y_2)$ if $x_1 \preceq_1 x_2$ and $y_2 \preceq_2 y_1$, $(x_1, y_1) \preceq_k (x_2, y_2)$ if $x_1 \preceq_1 x_2$ and $y_1 \preceq_2 y_2$. Negation can be defined as $\neg(x, y) = (y, x)$ if $L_1 = L_2$. As we will see later, the bilattice used in our system is constructed in this way. We will expand on this later.

5.2 Any-world Assumption

Non-monotonic logics allow a conclusion to be drawn on incomplete information. One way of doing such reasoning is to complete the missing part by *assumptions*. Taking into account assumptions means assigning truth values, implicitly or explicitly, to the unknown facts. The assumptions are usually based on the estimated states of the facts.

One of the most common assumptions is the *Closed World Assumption* (CWA). It assumes the default truth states of atoms to be $false$, therefore any atoms that cannot be proved to be $true$ are taken as $false$. Another well-known assumption is the *Open World Assumption* (OWA). OWA is a more cautious assumption in the sense that it assumes the default truth states of atoms to be $unknown$. Therefore, any atoms that cannot be proved to be $true$ are taken as $unknown$. However it also gives us less useful conclusions.

Using only one of these assumptions to represent the world uniformly is usually not appropriate. It is common that at the same time, some things can be safely assumed $false$ and others cannot. It is desirable to combine the two assumptions to form a new non-uniform assumption. More generally, we do not even want to assume something is absolutely $false$ or we have absolutely no knowledge in many cases. For example, we may want to assume that certain facts are "possibly false". This leads to the *Any-World Assumption* (AWA) [32], which was proposed in 2005. AWA gives us the power to form a large variety of assumptions on the possible truth of the atoms.

AWA unifies and extends the CWA and OWA by taking truth values from an arbitrary bilattice truth space and allow the default value of an atom to be any one of them. If in the assumptions, all the atoms are assigned to $false$, then it becomes CWA which says everything that cannot be inferred is false. If in the assumptions, all the atoms are assigned to $unknown$, then it becomes OWA which says everything cannot be inferred is unknown. The advantages are obvious: the truth, incompleteness and uncertainty can be represented in a finer granularity according to the experience and background information, therefore the assumptions we make carry more knowledge than before which in turn leads to more informed conclusions. The assumptions can be non-uniform which means the default truth values can vary for different atoms. This allows us to form more realistic assumptions.

The principle underlying AWA is to regard assumptions as an additional source of default information to complete the implicit knowledge provided by a logic program. Assumptions over a bilattice truth space provide default truth values for atoms. In order to minimise the impact brought by guesswork, assumptions are only used as the last resort. Only atom whose truth value cannot be inferred from the program is assigned to the default value given by the assumptions.

6 Shinren

6.1 Bilattice $\mathcal{NINE}$

As introduced in Section 5.1, a standard way of constructing an interlaced bilattice is by combining two lattices. The bilattice we employ, $\mathcal{NINE}$, is also built in this standard way. $\mathcal{NINE}$ is obtained by combining two identical lattices $L_1 = L_2 = \langle \{0, \frac{1}{2}, 1\}, \leq \rangle$ where $\leq$ is "less than or equal". The structure of $\mathcal{NINE}$ is shown in Figure 2. The truth values are represented as tuples (x, y) where $x, y \in \{0, \frac{1}{2}, 1\}$. The two orderings, $\preceq_t, \preceq_k$ are defined as:

$$(x_1, y_1) \preceq_t (x_2, y_2) \text{ if } x_1 \leq x_2 \text{ and } y_2 \leq y_1$$
$$(x_1, y_1) \preceq_k (x_2, y_2) \text{ if } x_1 \leq x_2 \text{ and } y_1 \leq y_2$$

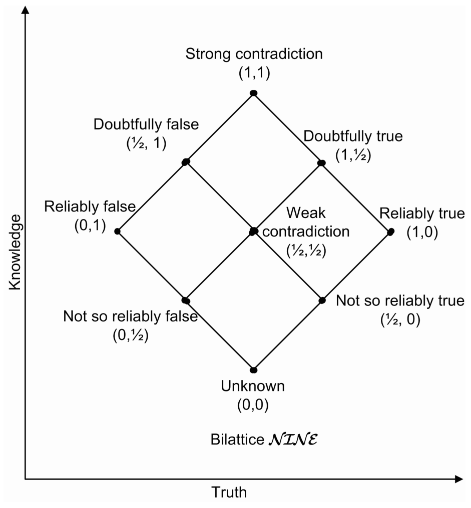

Fig. 2. The bilattice NINE

Given a statement with a truth value of the form (x, y), the intuitive meaning of the truth value is that x represents how much the statement is true (or you believe it is true), and y represents how much the statement is false (or you believe it is false). Let us elaborate on the meanings:

- (1,0) – **reliably true**: This value is given to a statement supported by very strong and reliable evidence. The possibility of the statement is actually false can be neglected.
- (0,0) – **unknown**: This value is given to a statement that for which have no information or are unable to verify (or falsify).
- (0,1) – **reliably false**: This value is given to a statement that is opposed by very strong and reliable evidence.
- ($\frac{1}{2}$,0) – **not so reliably true**: This value is given to a statement that is supported by weak or unreliable evidence. The statement could be false.
- (0,$\frac{1}{2}$) – **not so reliably false**: This value is given to a statement that is opposed by weak or unreliable evidence. We are not quite sure about its falsity.
- (1,$\frac{1}{2}$) – **doubtfully true**: This value is given to a statement that is supported by strong and reliable evidence, but for which we can also find some weak or unreliable evidence that opposes it. This usually happens when we combine information from different sources.
- ($\frac{1}{2}$,1) – **doubtfully false**: This value is given to a statement that is opposed by strong and reliable evidence, but for which we can also find some weak or unreliable evidence that supports it.
- ($\frac{1}{2}$, $\frac{1}{2}$) – **weak contradiction**: This value is given to a statement that is both supported by and opposed by weak or unreliable evidence from different sources.
- (1,1) – **strong contradiction**: This value is given to a statement that is both supported by and opposed by strong and reliable evidence from different sources.

From the above it is easy to understand the two orderings. For example, a statement which is reliably true contains more truth (or is more likely to be true) than a statement which is not so reliably true, i.e. $(\frac{1}{2}, 0) \preceq_t (1, 0)$. On the other hand, a reliably true statement gives us more information than a not so reliably true statement, i.e. $(\frac{1}{2}, 0) \preceq_k (1, 0)$. It is possible to extend the bilattice to a finer model of reliability or uncertainty. For example, using a lattice with the value domain $\{0, \frac{1}{3}, \frac{2}{3}, 1\}$, we can create a bilattice with 16 truth values that can represent more reliability levels. However, we do not do so for two reasons: first, enlarging the bilattice also increases the computational complexity. With enough expressiveness, we would like to avoid unnecessary cost; second, things like reliability and uncertainty cannot typically be measured precisely. There are no metrics and instruments we can use to standardise the measurement. A finer scale does not help in solving this problem, even worse, it may bring a false sense of precision. For these reasons, we stay with this basic form and extend it when it is necessary and possible.

Let us also explain the rationale behind this multi-valued truth space. Classical logic, which is the basis of many trust management systems, is bivalent, i.e. the only possible truth values are $true$ and $false$. It gives rise to "black and white thinking" where every proposition must be ascribed to "absolutely true" or "absolutely false". However, in the real world, many would agree with the statement "the only certainty is nothing is certain"[3]. Because classical logic lacks the ability of coping with the uncertainty in truth, mainstream trust management systems restrict the information that can be taken into account to "credentials". A credential is a statement signed by an issuer containing certain information about the credential holder and is believed to be highly reliable. The problems with credentials are two-fold: first, credentials are not able to carry every bit of information about the holder. We may find that signed information is just a very small fraction of all the information we can get. Second, in practice we do not encode negative information about the holder in credentials. The reason is simple: no one bothers to ask for a credential which is useless or has a negative effect to him. Again, we usually recognise a rogue merchant not from a "rogue merchant" credential signed by a government agency, but from various other sources like reviews in internet forums. If we want a more complete view of the trustee, using only credentials is not sufficient. We need to consider more information, possibly even that from the sources which are not so reliable. The multi-valued truth space gives Shinren the ability to represent and differentiate information with different qualities. And makes it possible for Shinren to utilise unreliable information.

The meet and join operators in terms of both orderings and the negation operator are then defined as follows:

$$\begin{aligned}
(x_1, y_1) \vee (x_2, y_2) &= (max(x_1, x_2), min(y_1, y_2)) \\
(x_1, y_1) \wedge (x_2, y_2) &= (min(x_1, x_2), max(y_1, y_2)) \\
(x_1, y_1) \oplus (x_2, y_2) &= (max(x_1, x_2), max(y_1, y_2)) \\
(x_1, y_1) \otimes (x_2, y_2) &= (min(x_1, x_2), min(y_1, y_2)) \\
\neg(x, y) &= (y, x)
\end{aligned}$$

We will explain these with some examples. Given a statement p which is reliably true and q that is not so reliably true, the truth value of their conjunction is $p \wedge q =$

[3] Pliny the Elder, Roman scholar (23-79 AD).

$(1, 0) \wedge (\frac{1}{2}, 0) = (min(1, \frac{1}{2}), max(0, 0)) = (\frac{1}{2}, 0)$, i.e. not so reliably true. This is easy to understand. Let p be "Alice is a student" and q be "Alice is a research assistant", then the statement "Alice is both a student and a research assistant" cannot be very reliable because we are not quite sure about the fact that she is a research assistant. Consider another example: in the court of a murder case, the prosecutor submits a CCTV record as evidence showing that the suspect was at the crime scene when the murder was happening, while the counsel of the suspect has a witness, who is a friend of the suspect, to certify that the suspect was in a pub 50 km away from the scene at the same time. It turns out the conclusion of whether the suspect was at the scene after we combine these two pieces of evidence is: $(1, 0) \oplus (0, \frac{1}{2}) = (max(1, 0), max(0, \frac{1}{2}) = (1, \frac{1}{2})$. That is, although doubtful, we would believe the suspect is at the scene. The reason is that the video record is more reliable evidence.

6.2 Shinren Policy Language

The syntax of Shinren is based on the logic programming language Datalog [33], with certain extensions. The alphabet consists of the following classes of symbols:

1. *Variables*, written as strings starting with a capital letter. For example, X, $Name$.
2. *Constants*, written as strings not starting with a capital letter. For example $alice$, $student$. The constants set contains several disjoint subsets: a set of entity constants, a set of trust scopes, a set of truth values and a set of other application specific constants. A trust scope is what the trustor wants to achieve by relying on the trustee or how the trustee is expected to behave. For example "be a good car mechanic" or "to read my document".
3. *Predicate symbols*, symbols used to denote properties of objects or relations between objects. We will give more details on predicate symbols later.

As in Datalog, we do not have function symbols. The restriction is necessary to ensure finiteness of models and termination of inference. As usual, a *term* is either a variable or a constant. We use *entity term* to refer to a term which can be either an entity constant or a variable, and similar with the others. If p is a n-ary predicate symbol and $t_1, ..., t_n$ are terms, then $p(t_1, ..., t_n)$ is an *atomic formula* or simply *atom*. A atom $p(t_1, ..., t_n)$ is *ground* if $t_1, ..., t_n$ are constants. A *literal* is an atom or the negation of an atom. Furthermore, a positive literal is an atom and a negative literal is the negation of an atom. A *consensus formula* is of the form of $L_1 \otimes ... \otimes L_n$, where L_i is a literal. A *gullibility formula* is of the form of $L_1 \oplus ... \oplus L_n$, where L_i is a literal. A rule, or policy, is of the form:

$$A :- \varphi_1, ..., \varphi_n.$$

where A is an atom and each φ_i is a literal, a consensus formula or a gullibility formula. "$: -$" is taken as "$\leftarrow$" and "," is taken as "$\wedge$". The atom A on the left-hand side of the rule is called its *head* and the conjunction $\varphi_1, ..., \varphi_n$ on the right-hand side is called its *body*. Certain types of rules may also have a priority label $\langle lab \rangle$ attached before the rules (will explain later). An *assertion* is a special type of rule defined as:

$$A :- b.$$

Where A is a ground atom and b is a truth value. An assertion can be understood as A has a truth value b. A *fact set* is a finite set of assertions. An *assumption set* is also a finite set of assertions. The difference is that the fact set contains the real truth values for the atoms while the assumption set contains the assumptions, i.e. assertions about the default values of the atoms. The assumptions are used only when no facts about the atoms can be found in the fact set or be inferred. We do not need to explicitly represent assumptions of the form $A :- (0, 0)$. If no assumption about an atom can be found in the assumption set, the default value is $(0, 0)$. A *program* is the union of a finite set of rules, a fact set and an assumption set.

We divide the predicates symbols into two sets: extensional database (EDB) predicates and intensional database (IDB) predicates. The EDB predicates represent the evidence or facts that can be used in reasoning about trust. The IDB predicates are relations. The EDB predicates can appear in a rule's body but not in the head. We use the following EDB predicates in our trust policy language. In the upcoming text, we abuse the notation a little bit by enclosing the optional arguments of predicates in square brackets. For example, $p(t_1, t_2[, ..., t_n])$ means the predicate symbol p is "overloaded" with the number of arguments varying from 2 to n.

– A set of *credential* predicates. This is a set of predicates $cred\_name(t_1, t_2[, ..., t_n])$ where $cred\_name$ is the name of the credential, t_1, t_2 are entity terms and $t_3, ..., t_n$ are attribute values. A credential predicate can be understood as "a credential of the name $cred\_name$ is issued by issuer t_1 to an entity t_2 and contains the following attribute values $t_3, ..., t_n$". A credential may not contain any attribute value, in this case the predicate for this credential is simply $cred\_name(t_1, t_2)$. We assume the existence of a standard ontology and a standard format for the credentials so that they can be represented correctly. For example, $student(xu, alice, ug)$ represents a student credential signed by X University to Alice, which also certifies that she is classified as an undergraduate.
– A set of predicates $recommendation(t_1, t_2, t_3[, ..., t_n])$, where t_1, t_2 are entity terms, t_3 is a trust scope term and $t_4, ..., t_n$ are other parameters. This type of predicate is used to represent the recommendations, i.e. the trust opinions, which can be sought from peers. The predicate reads as "the recommender t_1 recommends t_2 for the scope t_3, with regard to $t_4, ..., t_n$". For example, $recommendation(alice, bob, repairCar)$. We do not explicitly have another predicate to represent negative recommendations. This type of recommendation is captured by an assertion with a negative truth value b, i.e. $b \prec_t (0, 0)$.
– A predicate $reputation(t_1, t_2, t_3, t_4)$ where t_1 is a entity term and t_2 is a trust scope. Reputation can be viewed as the aggregation of trust opinions from a community. This predicate represents an entity t_1's reputation in terms of a scope t_2, which is provided by a reputation system t_3 with a value of t_4.
– A set of *local knowledge* predicates. This set of predicates represents any other information that can be gathered by the system. For example, the behaviour patterns of past interactions (if any), unsigned statements and so on. These can be used to provide a continuous monitoring and feedback mechanism, so that the system can use direct experience in trust evaluation.

The IDB predicates include:

– A set of trust predicates $trust(t_1, t_2[, ..., t_n])$ where t_1 is a entity term, t_2 is a trust scope and $t_3, ..., t_n$ are other parameters of the intended trust relationship. Such a predicate can be read as “the system trusts the trustee t_1 for the scope t_2, with regard to $t_3, ..., t_n$”.

– A set of distrust predicates $distrust(t_1, t_2[, ..., t_n])$ which are the counterparts of the trust predicates. The predicate reads as “the system distrust the trustee t_1 for the scope t_2, with regard to $t_3, ..., t_n$”. Using $distrust$ predicates enables policy makers to specify explicitly in which situations the trustee should not be trusted. It is similar to the concept of “explicit deny” in access control systems.
– A set of *constraint* predicates. Constraints are conditions on attribute values and are useful in specifying conditions such as “age greater than 21”, “reputation no less than 0.7”. The constraint predicates are special in the sense that they are not defined in the program, i.e. not in any rule’s head, and their semantics depends on a constraint solver and a constraint domain. We employ a constraint solver as a black box component and assume a tractable constraints domain. We also require that each constraint used in a policy must be linked to another non-constraint predicate (see section 6.3).
– A set of application-specific predicates. These predicates capture the other possible relationships existing in the system. They can be defined by policy makers when necessary.

By using the Shinren trust policy language, policy makers can define both trust policies and distrust policies, i.e. rules whose heads are $trust$ or $distrust$ predicates. They can also label the policies with *priority levels*. The priority levels express how preferable a policy is. The priority levels in Shinren language are defined as a finite set of non-negative integers $\{0, 1, ..., n\}$. 0 is reserved for default assignment rules. The higher the number is, the higher the priority is. For each priority level, policy makers also define two thresholds in terms of $\preceq_t$ or $\preceq_k$ or both, one for distrust policies and one for trust policies. The thresholds are used to filter poor answers. Answers that satisfy the threshold are called *admissible answers*. Note that only trust or distrust policies need labels, the other policies are not prioritised. When the system is asked to evaluate trust, it starts from policies with the highest priority. Distrust policies are evaluated before trust policies. In other words, distrust policies have a higher priority than trust policies at the same priority level. If an admissible answer can be found, then the evaluation ends. Otherwise it continues to evaluate the trust policies at the same level. If there are still no admissible answers, the system continues with the policies at the next level. When an admissible answer is found with truth value b, an answer for its counterpart is asserted with a truth value $\neg b$. For example, if the evaluation ends with an admissible answer $distrust(\boldsymbol{a}) = (\frac{1}{2}, 0)$, we also have $trust(\boldsymbol{a}) = (0, \frac{1}{2})$. If after evaluating all the policies at higher priority levels, an admissible answer is still not found, the default value is applied. The default value assignment rules may be omitted, in this case the default value is $(0, 0)$.

The prioritisation mechanism can be used to resolve modality conflicts introduced by trust and distrust policies. Trust and distrust are semantically opposite and it is possible in some situations that both are true based on the policies. Therefore we need to handle the possible conflicts. With priority levels, the conflicts can be resolved by “interlacing” distrust and trust policies and the decisions are governed by the policies with the highest priority levels which give admissible answers. The priority levels can also be used to order trust decisions. For example, if we have decided both Alice and Bob can be reliably trusted, we may prefer Alice if the decision about her came from a trust policy with a higher priority level, i.e. a more preferable policy. The truth values and priority levels can give hints to the decision maker. If the decision is not reliable or from a less preferable policy, it may indicate that the decision is not favourable and may be risky. The decision maker can activate some compensative controls based on the truth value and priority levels.

In order to achieve prioritisation, we also need to pose a syntactical restriction on the policies in order to guarantee the policies can be evaluated correctly. Simply speaking, we require that there are no cyclical dependencies between ground atoms. Formally, let P be a program and P^{inst} be the collection of all the ground instances of P. For any ground atom $A(\boldsymbol{a})$ and $B(\boldsymbol{b})$, we say $A(\boldsymbol{a})$ depends on $B(\boldsymbol{b})$ if there is a ground rule in P^{inst} such that $A(\boldsymbol{a})$ is in the head and $B(\boldsymbol{b})$ or $\neg B(\boldsymbol{b})$ is in the body or any of the atoms in the body depends on $B(\boldsymbol{b})$. Particularly, we treat $trust(\boldsymbol{a})$ and $distrust(\boldsymbol{a})$ as one atom because they are always derived together. Therefore, if $A(\boldsymbol{a})$ depends on $trust(\boldsymbol{b})$ or $distrust(\boldsymbol{b})$, it also depends on $distrust(\boldsymbol{b})$ or $trust(\boldsymbol{b})$; if $trust(\boldsymbol{a})$ or $distrust(\boldsymbol{a})$ depends on $B(\boldsymbol{b})$, $distrust(\boldsymbol{a})$ or $trust(\boldsymbol{a})$ also depends on $B(\boldsymbol{b})$. P^{inst} must be able to be stratified as several disjoint strata $P_0, ..., P_n$ such that $P^{inst} = P_0 \cup ... \cup P_n$ and for every ground atom $A(\boldsymbol{a})$ which is defined in P_i and depends on $B(\boldsymbol{b})$, the definition of $B(\boldsymbol{b})$ can be found in $P_0 \cup ... \cup P_{i-1}$.

6.3 Semantics

We first show how the semantics of logic programming can be extended from the classical 2-valued space to a bilattice. Let $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ be a bilattice. The *Interpretation* of a logic program on the bilattice is a mapping I from ground atoms to members of $\mathcal{B}$. $\mathcal{B}$ is the set of all the truth values, i.e. the truth space. An interpretation can be extended from atoms to formulae in the usual way:

– for $b \in \mathcal{B}$, $I(b) = b$;
– for formulae φ and φ', $I(\varphi \wedge \varphi') = I(\varphi) \wedge I(\varphi')$, and similarly for $\vee, \otimes, \oplus, \neg$;
– $I(\exists x \varphi(x)) = \bigvee\{I(\varphi(t)) : t\ is\ a\ ground\ term\}$;
– $I(\forall x \varphi(x)) = \bigwedge\{I(\varphi(t)) : t\ is\ a\ ground\ term\}$

Here we need to explain a little about the interpretation of constraints. In Shinren, as in many policy systems using constraints (e.g. [34]), constraints can be seen as a kind of plug-in. Constraint solving is not dealt internally by Shinren, but is delegated to an external constraint solver. The constraint solver is given a set of constraints and decides whether the constraints are satisfied, then returns an answer. The answer may affect the final result of policy evaluation. The benefits of using constraint solvers as external black boxes are obvious: we do not need to consider the implementation details of constraint solvers and we do not need to change the semantics of the policy language when dealing with different constraint domains.

A constraint domain, loosely speaking, is a set of constraints with an interpretation that defines the validity of constraints. Often a constraint domain is required to satisfy certain properties, e.g. contains equality predicates, closed under variable renaming and so on. In the following discussion, we assume a well-defined and tractable constraint domain, denoted by $\mathcal{C}$. All the constraints used in the policy bodies are taken from $\mathcal{C}$. Also, to distinguish the constraints from the other predicates defined in Shinren, we use $c_1, ...c_n$ to denote constraint predicate symbols defined in $\mathcal{C}$ and $p_1, ...p_n$ to denote the other predicate symbols. Given a policy of the form:

$$A :\!-\, p_1, ...p_n, c_1, ...c_m.$$

We say c_i is linked to p_j if there exists a term t such that t is an argument of both c_i and p_j. For example, in the policy:

$$trust(X, bid, Item) :\!-\, soldBy(X, Item), itemPrice(Item, Price), Price \leq 20.$$

the constraint $Price \leq 20$ is linked to predicate $itemPrice(Item, Price)$. Each constraint must be linked to at least one non-constraint predicate in the policy. In other words, we do not want to set any irrelevant constraints in the policies. We also define that for a ground constraint c_i in a policy $A :\!-\, p_1, ...p_n, c_1, ...c_m$:

$$I(c_i) = \begin{cases} I(\phi) & \text{if} \models_{\mathcal{C}} c_i \\ \neg I(\phi) & \text{otherwise} \end{cases}$$

where $\phi = \bigwedge\{p_j : c_i \text{ is linked to } p_j, j \in [1, n]\}$. Loosely speaking, this means that a constraint's interpretation in the policy depends on its validity in its constraint domain and also its linked predicates.

The semantics of Shinren is an extension of the generalised Kripke-Kleene Semantics [29, 35]. A program P is first transformed into P^* in the following way:

1. put in P^* all ground instances of rules and facts (over the Herbrand universe).
2. replace all the ground unlabelled rules in P^* which have the same head, $A :\!-\varphi_1,\ A :\!-\varphi_2, \ldots$ with one rule $A :\!-\varphi_1 \vee \varphi_2 \vee \ldots$.
3. replace all the ground labelled rules in P^* which have the same head and label, $\langle lab \rangle\ A :\!-\varphi_1, \langle lab \rangle\ A :\!-\varphi_2, \ldots$ with one rule $\langle lab \rangle\ A :\!-\varphi_1 \vee \varphi_2 \vee \ldots$.
4. if a non-constraint ground atom A is not head of any rule in P^*, then the rule $A :\!-H(A)$ is added to P^*, where $H(A)$ is the default truth value for A in the assumption set.

P^* can then be partitioned as $P_0^*, ..., P_n^*$ as stated in Section 6.2. Then M, the unique minimal model after resolving the conflicts, is constructed iteratively:

$$M_0 = \Phi_{P_0^*} \uparrow^{\omega}$$
$$M_i = \Phi_{P_i^*} \uparrow^{\omega} (I_{i-1}), 1 \leq i \leq n$$
$$M = I_n$$

where $\Phi_{P_i^*}$ is the immediate consequence operator which is defined as $\Phi_{P_i^*}(I)(A) = I(\phi)$ for all $A :\!-\phi \in P_i^*$, M_i is the fixpoint obtained by applying $\Phi_{P_i^*}$ to I_{i-1} and I_i is defined as follows: $I_0 = M_0$, for $1 \leq i \leq n$, put into I_i the content of I_{i-1} and any valuations in M_i for atoms which are not $trust$ or $distrust$, for any pair of atoms $trust(\boldsymbol{a})$ and $distrust(\boldsymbol{a})$, put into I_i the admissible answer in M_i derived from the policy with highest priority and also its counter part with negated truth value. Intuitively, M is a model for a program obtained from P^* which removes all the trust and distrust policies from P^* except those that give admissible answers with highest priorities, i.e. the preferred program. M is unique and minimal with respect to $\preceq_k$. Detailed theorems and proofs can be found in Appebdix A.

7 Example: Electronic Marketplace

Alice is a big fan of Internet shopping and she often visits a website called tBay which is an electronic marketplace like e-bay. Although she has bought a lot of items with very low prices, she also had several unpleasant experiences. So she wants to be cautious before she bids on anything from the website. She decides that she will only bid on items from sellers who live in the UK, have been registered no less than 6 months and have at least 80% positive feedback. She will also ask her friend Bob about his opinion and will not consider a seller if Bob does not like him. However, she knows tBay has a special procedure for items with bid prices lower than £20: in case of dispute, tBay will fully refund the buyer. Since she is not going to lose money, Alice is willing to bid in such situations regardless of her other constraints above. But Alice also has a more important principle: she will never trade with someone who has cheated her. She has a blacklist of such sellers. Alice's policies are:

$\langle 3\rangle\ distrust(X, bid, Item) :\!-\ inBlackList(X).$

$\langle 2\rangle\ trust(X, bid, Item) :\!-\ soldBy(X, Item), itemPrice(Item, Price), Price \leq 20.$

$\langle 1\rangle\ distrust(X, bid, Item) :\!-\ \neg recommendation(bob, X, bid, Item).$

$\langle 1\rangle\ trust(X, bid, Item) :\!-\ seller(tBay, X, Location, RegisterPeriod),$
$\quad Location = uk, RegisterPeriod \geq 6, soldBy(X, Item),$
$\quad reputation(X, goodSeller, tBay, Y), Y \geq 0.8.$

Alice's policies have 3 priority levels. At the highest level is the policy which should not be overridden by any other policies. At the second level is a trust policy that allows her to interact with any seller when there is no risk. The lowest level has two policies for general cases. In the policies, $seller(tBay, X, Location, RegisterPeriod)$ represents a seller credential signed by tBay. $soldBy(X, Item)$, $itemPrice(Item, Price)$ and $inBlackList(X)$ are local knowledge predicates supplying useful information. For each priority level, Alice defines thresholds for admissible answers to be $(0, 0) \prec_t$, which means only answers somehow true (reliably true, not so reliably true, doubtfully true) will be admissible.

Along with the policies, Alice also has a set of assumptions:

$soldBy(X, Item) :\!-\ (0, 1).$
$itemPrice(Item, Price) :\!-\ (0, 1).$
$inBlackList(X) :\!-\ (0, 1).$

Recall that $(0, 1)$ means "reliably false". Alice's assumptions are: if she cannot find any information that says an item is sold by seller X, then this item is not sold by X; if she cannot find any information that says an item is sold for a certain price, then it is not sold for this price; if she cannot find a seller in her blacklist, then he is not in her blacklist. These are easy to understand. All the other predicates are left with default values of $(0, 0)$, i.e. unknown. Different default values may make a big difference. For example, if Alice assumes $recommendation(bob, X, bid, Item)$ to be false, i.e. add $recommendation(bob, X, bid, Item) :\!-\ (0, 1)$ to her assumption set, then she cannot bid anything with a price higher than £20 when she cannot contact Bob. In such cases, since she cannot get recommendations from Bob, the default value will be used and the policy

$$distrust(X, bid, Item) :\!-\neg recommendation(bob, X, bid, Item).$$

will always give results of "distrust" with truth value $(1, 0)$.

More complicated policies are also possible. For example, if Alice has another policy which says she will bid if at least two of her friends recommend the seller. This can be written as:

$$\begin{aligned}&trust(X, bid, Item) :\!- friend(F1), friend(F2),\\&\quad recommendation(F1, X, bid, Item) \otimes recommendation(F2, X, bid, Item).\end{aligned}$$

Alice collects the following facts when she tries to find a cheap iPod on tBay:

$$\begin{aligned}&soldBy(carol, ipod) :\!- (1, 0).\\&seller(tBay, carol, uk, 12) :\!- (1, 0).\\&reputation(carol, goodSeller, tBay, 0.9) :\!- (\frac{1}{2}, 0).\\&itemPrice(ipod, 80) :\!- (1, 0).\end{aligned}$$

Although not signed, Alice considers the information about who is the seller and the price of the item as reliable. However, the reputation is not. Alice knows at least ten ways which sellers can boost their reputation quickly.

When evaluating the policies, only the assumptions:

$$\begin{aligned}&recommendation(bob, carol, bid, ipod) :\!- (0, 0)\\&inBlackList(carol) :\!- (0, 1)\end{aligned}$$

are used. This is because Alice does not have any relevant information. The other assumptions are not used because Alice has collected the facts and therefore does not need to assume anything.

Let us also explain how the trust (distrust) policies are evaluated. Shinren starts from priority level 3. For the distrust policy at this level, the body is $inBlackList(carol)$ with truth value $(0, 1)$ in the interpretation. Therefore $distrust(carol, bid, ipod)$ is evaluated to be $(0, 1)$, according to this policy. Because $(0, 0) \not<_t (0, 1)$, this answer is not admissible and is discarded. The policy with priority 2 does not have an admissible answer either. Given $itemPrice(ipod, 80) = (1, 0)$, the constraint $Price \leq 20$ is not satisfied because the price is £80. This constraint is linked to $itemPrice(ipod, 80)$, so its truth value is $\neg(1, 0) = (0, 1)$. Overall, $trust(carol, bid, ipod)$ is evaluated to $(0, 1)$ according to this policy. The answer is also discarded. Because Alice cannot get a recommendation from Bob, the default value is used and the distrust policy at priority level 1 is evaluated to $(0, 0)$. The answer is also not admissible. The last policy is evaluated to $(\frac{1}{2}, 0)$ and therefore is admissible. Because it is a trust policy, we add $trust(carol, bid, ipod) = (\frac{1}{2}, 0)$ and also $distrust(carol, bid, ipod) = (0, \frac{1}{2})$ to the model. Alice now knows that although Carol can be trusted, she might still be cheated.

8 Example: Healthcare in the Community

Dr Taylor runs a medical clinic in a small town. An unconscious patient is brought to the clinic. From the driver's licence, Dr Taylor learns that the patient is called Mr Johnson. Mr Johnson is a tourist and stayed in a local hotel before he was brought here. The owner of the hotel, who brought Mr Johnson in, tells Dr Taylor that the patient experienced breathing difficulties during breakfast and then passed out a few minutes later. Dr Taylor examines the patient's trachea and hears the lung sound. He decides to intubate the patient in order to let air pass freely to and from the lungs. The patient's temperature is normal and the results of a blood test show no signs of infection. Blood pressure and heart rate are also normal. Dr Taylor decides to check the patient's medical history in order to see whether the symptoms were caused by drugs or allergies. From his computer, Dr Taylor sends a request to the Smith GP practice, found in documents in Mr Johnson's wallet.

The electronic medical record system of the Smith GP practice uses the Shinren trust management system to control who can access patients' medical histories. The policies which regulate the access to a patient's medical history are shown blow:

$\langle 3\rangle\ distrust(X, read, med\_history, Y) :- \neg doctor(bma, X).$

$\langle 3\rangle\ trust(X, read, med\_history, Y) :- consent(Y, X, read, med\_history)$

$\langle 3\rangle\ trust(X, read, med\_history, Y) :- agent(Y, Z), consent(Z, X, read, med\_history)$

$\langle 2\rangle\ trust(X, read, med\_history, Y) :- answer(X, DOB, ADDRESS),$
$\quad personal\_info(Y, DOB2, ADDRESS2), DOB = DOB2,$
$\quad ADDRESS = ADDRESS2.$

$\langle 1\rangle\ trust(X, read, med\_history, Y) :- collocated(X, Y).$

Patients' medical histories are sensitive and should only be revealed to doctors who are treating the patients. The distrust policy at level 3 says that X is not allowed to read patient Y's medical history if X does not have a doctor credential signed by the BMA (British Medical Association). The second trust policy at the same level says X is trusted to read patient Y's medical history if Y gives his consent. However, in real-life, it is not always possible to get the patient's consent, e.g. in the case that the patient is in coma. Then a third party consent from the patient's agent, usually the next of kin, also has the same effect. In emergency situations where no consent can be obtained, it is necessary to verify that the doctor is indeed treating the patient before letting the doctor access the information without consent. For example, the verification might be done by letting the doctor provide the patient's personal information and comparing it with the data stored, or using a location service to verify that the doctor is co-located with the patient. Accesses without consent are logged and audited.

Dr Taylor provides his doctor credential and also supplies information about Mr Johnson's birthday and address correctly. The access is granted and logged. Alas the medical history does not provide too much useful information. At the same time, Mr Johnson's condition becomes worse. He starts to have seizures and EEG (electroencephalogram) shows abnormal brain activities.

Dr Taylor suspects that the problem may be in Mr Johnson's brain. However, he is not a neurologist and needs someone to help in diagnosing the patient. Dr Taylor starts looking for help. He searches the NHS database using Shinren with the policies shown below:

$$\langle 2\rangle\ trust(X, specialist, neurology) :\!-consultant(Hos, X, neurology),$$
$$hospital(NHS, Hos), member(aon, X, Level), Level >= 2.$$
$$\langle 1\rangle\ trust(X, specialist, neurology) :\!-consultant(Hos, X, neurology),$$
$$hospital(NHS, Hos), member(aon, X, 1), member(aon, Y, Level), Level >= 2,$$
$$recommendation(Y, X, specialist, neurology).$$

The first policy says that Dr Taylor will trust X as a specialist in neurology if X has a consultant credential signed by an NHS hospital which states that X is a consultant in neurology. X must also be a member of the Association of Neurologists with level no lower than senior member. The second policy says almost the same except that if the level of X in the Association of Neurologists is not high enough, he needs a recommendation from a senior member or higher.

Dr Taylor finds 20 doctors who fit his requirements. Among them, he selects Dr Ford, a senior member of the Association of Neurologists who works for Victoria Hospital. Dr Ford is also willing to offer assistance. Dr Taylor sets up a video conference with Dr Ford. After hearing the observations and checking the examination results, Dr Ford suggests that the problem could be caused by a clot in the patient's brain. However, a brain tumour also fits the symptoms. The diagnosis can be confirmed by an MRI (Magnetic Resonance Imaging) scan or a brain biopsy. However, the clinic does not have the equipment and the patient's condition is not suitable for transportation. Dr Ford then suggests that in this situation, Dr Taylor should immediately treat the patient with tPA (tissue Plasminogen Activator), a medicine which helps resolving blood clots, because a long delay could cost the patient's life. If the patient's condition gets better, then the diagnosis of a blood clot can be confirmed, otherwise it suggests a brain tumour.

Dr Ford's plan could be quite dangerous. So Dr Taylor wants to hear a second opinion. To ensure the opinion is independent and fair, Dr Taylor adds another policy before he searches for the second specialist. The policy rules out all the specialists working in the same hospital as Dr Ford.

$$\langle 2\rangle\ distrust(X, specialist, neurology) :\!-\ consultant(victoria, X, neurology).$$

This time Dr Taylor finds Dr Grant, a senior member of the Association of Neurologists who works for the Albert Hospital. Dr Grant confirms that there is no better solution in this situation. Dr Taylor starts to treat the patient with tPA, and watches him closely. 24 hours later, the patient wakes up. After the patient's condition is stabilised, he is transferred to the nearest major hospital for further diagnosis and treatment.

9 Implementation

We have implemented a prototype of Shinren. As shown in Figure 3, Shiren consists of five major modules. Among the five modules, the credential module, the recommendation module, the reputation module and the state module are responsible for retrieving and interpreting information from different sources, and the policy interpreter module is responsible for making decisions according to the policies and the information gathered.

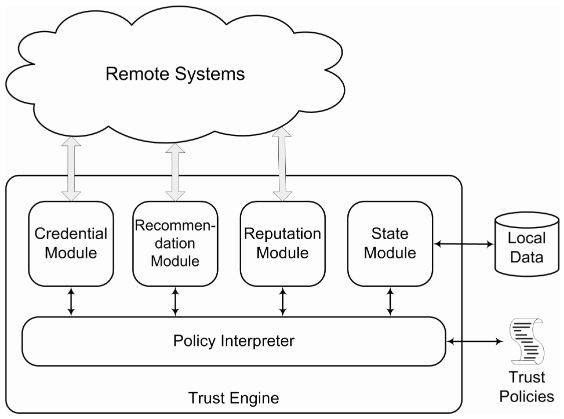

Fig. 3. Shinren Trust Engine

The Shinren prototype is implemented in Java 1.5. The policy interpreter evaluates queries in a bottom-up fashion as in many other datalog-based systems. Policies are loaded into the policy interpreter as plain text files. The rules are stratified when they are loaded by analysing the predicate dependency relationships. To answer a query, the interpreter first initialises an interpretation which is an instance of the Assertion-Set class. The interpreter queries the other four modules in order to gather facts, i.e. ground instances of the predicates with truth values, which are needed for policy evaluation. The facts are stored in tables related to the predicates in the interpretation. After the interpreter obtains all facts, it constructs the Herbrand universe by collecting all the constants from the query, rules and facts. The interpreter then puts into the initial interpretation assumptions for all the other ground atoms which are in the Herbrand base. It then starts evaluating policies iteratively from the lowest strata. Each rule in the stata is grounded with regard to the Herbrand base and then the interpreter applies the immediate consequence operator to each ground instance. The immediate consequence operator retrieves the truth values for the ground atoms in the rule body from the current interpretation and passes them through the evaluation tree of the rule to obtain the truth value of the ground head atom. The ground atom along with the truth value is a newly generated fact and the table for the head predicate in the interpretation is updated. If an entry with the same ground tuple is already in the table, the truth value of the old entry is ORed with the truth value of the new entry; otherwise the new entry is inserted into the table. Trust (distrust) policies in the same strata are evaluated sequentially by priority level until an admissible answer, i.e. an answer that satisfies the threshold defined for this level, is found. The evaluation of the strata ends when the interpretation does not change anymore. Then the interpreter evaluates the rules in the next strata. The evaluation of the query ends after the interpreter evaluates all the strata containing the rules with the queried atoms as heads.

A preliminary top down policy interpreter which is under developing can be found in Appendix B. However the correctness of this implementation has not been proved.

10 Conclusion and Future work

In this paper, we have presented Shinren, a novel non-monotonic trust management system based on bilattices and the any-world assumption. The syntax of the Shinren policy language is based on Datalog with certain extensions such as negation, constraints and prioritisation. The semantics extends the Kripke-Kleene semantics over bilattices. Shinren can utilise unreliable even contradictory information and supports prioritisation which resolves conflicts and provides decision support. We demonstrated the power of Shinren by two comprehensive examples and outlined its implementation.

One aspect that we would like to investigate further is prioritisation. The current prioritisation mechanism in Shinren is at the meta-level. It works but is not convenient in practice because it is external to the bilattice. However, prioritisation can also be viewed as another ordering. We would like to extend our bilattice, so that a third ordering could be integrated into the theory. This would make prioritisation a built-in feature.

AWA uses the concept of non-uniform assumption. However, its assumptions are static. We are interested in researching dynamic assumptions which would mean that changes in knowledge could lead to the change of the assumptions. Dynamic assumptions would enable a trust management system to generate more accurate conclusions according to the context. Previous works in belief revision [36] and dynamic prioritisation [37] are possible stepping stones in this direction.

Acknowledgments

This research was supported by the UK's EPSRC research grant EP/C537181/1 (Caregrid) and EU FP7 research grant 213339 (ALLOW).

References

1. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: IEEE Symposium on Security and Privacy, IEEE Computer Society (1996) 164–173
2. Mayer, R.C., Davis, J.H., Schoorman, D.F.: An integrative model of organizational trust. The Academy of Management Review **20**(3) (1995) 709–734
3. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The keynote trust-management system, version 2. RFC 2704 (1999)
4. Jim, T.: Sd3: A trust management system with certified evaluation. In: SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2001) 106–115
5. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: IEEE Symposium on Security and Privacy. (2002) 114–130

6. Hess, A., Seamons, K.E.: An access control model for dynamic client-side content. In: SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies, New York, NY, USA, ACM Press (2003) 207–216
7. Carbone, M., Nielsen, M., Sassone, V.: A formal model for trust in dynamic networks. In: SEFM. (2003) 54–61
8. Blaze, M., Feigenbaum, J., Keromytis, A.D.: The role of trust management in distributed systems security. In Vitek, J., Jensen, C.D., eds.: Secure Internet Programming. Volume 1603 of Lecture Notes in Computer Science., Springer (1999) 185–210
9. Rivest, R.L., Lampson, B.: SDSI - a simple distributed security infrastructure. "http://people.csail.mit.edu/rivest/sdsi10.html" (1996)
10. Ellison, C.: SPKI requirements. RFC 2692 (1999)
11. Gunter, C.A., Jim, T.: Policy-directed certificate retrieval. Softw., Pract. Exper. **30**(15) (2000) 1609–1640
12. Becker, M.Y., Sewell, P.: Cassandra: Distributed access control policies with tunable expressiveness. In: POLICY, IEEE Computer Society (2004) 159–168
13. Halpern, J.Y., van der Meyden, R.: A logic for sdsi's linked local name spaces. Journal of Computer Security **9**(1/2) (2001) 105–142
14. Li, N., Mitchell, J.C.: Datalog with constraints: A foundation for trust management languages. In Dahl, V., Wadler, P., eds.: PADL. Volume 2562 of Lecture Notes in Computer Science., Springer (2003) 58–73
15. Chu, Y.H., Feigenbaum, J., LaMacchia, B.A., Resnick, P., Strauss, M.: Referee: Trust management for web applications. Computer Networks **29**(8-13) (1997) 953–964
16. Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., Ravid, Y.: Access control meets public key infrastructure, or: Assigning roles to strangers. In: IEEE Symposium on Security and Privacy. (2000) 2–14
17. Czenko, M., Tran, H., Doumen, J., Etalle, S., Hartel, P., den Hartog, J.: Nonmonotonic trust management for P2P applications. Electronic Notes in Theoretical Computer Science **157**(3) (2006) 113–130
18. Marsh, S.P.: Formalising Trust as a Computational Concept. PhD thesis, University of Stirling (1994)
19. Jøsang, A.: A logic for uncertain probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **9**(3) (2001) 279–212
20. Yu, B., Singh, M.P.: Detecting deception in reputation management. In: AAMAS, ACM (2003) 73–80
21. Reiter, R.: On closed world data bases. In: Logic and Data Bases. (1977) 55–76
22. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artif. Intell. **13**(1-2) (1980) 27–39
23. Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1-2) (1980) 81–132
24. Kleene, S.C.: On notation for ordinal numbers. J. Symb. Log. **3**(4) (1938) 150–155
25. Grosof, B.N.: Courteous logic programs: Prioritized conflict handling for rules. Research Report RC 20836(92273), IBM (1997)
26. Ginsberg, M.L.: Multivalued logics: a uniform approach to reasoning in artificial intelligence. Computational Intelligence **4** (1988) 265–316
27. Shepherdson, J.C.: Negation as failure, completion and stratification. In Gabbay, D.M., Hogger, C.J., Robinson, J.A., eds.: Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 5. Oxford Science Publications (1998)
28. Clark, K.L.: Negation as failure. In: Logic and Data Bases. (1977) 293–322
29. Fitting, M.: A kripke-kleene semantics for logic programs. J. Log. Program. **2**(4) (1985) 295–312
30. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP. (1988) 1070–1080

31. Gelder, A.V., Ross, K.A., Schlipf, J.S.: Unfounded sets and well-founded semantics for general logic programs. In: PODS, ACM (1988) 221–230
32. Loyer, Y., Straccia, U.: Any-world assumptions in logic programming. Theor. Comput. Sci. **342**(2-3) (2005) 351–381
33. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. **1**(1) (1989) 146–166
34. Becker, M.Y., Sewell, P.: Cassandra: Flexible trust management, applied to electronic health records. In: CSFW, IEEE Computer Society (2004) 139–154
35. Fitting, M.: Bilattices and the semantics of logic programming. J. Log. Program. **11**(1&2) (1991) 91–116
36. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. J. Symb. Log. **50**(2) (1985) 510–530
37. Brewka, G.: Reasoning about priorities in default logic. In: AAAI. (1994) 940–945
38. Fitting, M.: Billatices are nice things. In Bolander, T., Hendricks, V.F., Pedersen, S.A., eds.: Self-reference. Number 178 in CSLI Lecture notes. CSLI Publications (2006) 53–77
39. Sterling, L., Shapiro, E.: The art of Prolog (2nd ed.): advanced programming techniques. MIT Press, Cambridge, MA, USA (1994)

A Theorems and Proofs

Definition 1 *A conflicting set w.r.t. P^* is a set of labelled rules in P^* such that the heads of the rules are the same ground atom $trust(\boldsymbol{a})$ (or its counterpart $distrust(\boldsymbol{a})$).*

A conflicting set contains all the rules regarding $trust(\boldsymbol{a})$ ($distrust(\boldsymbol{a})$) with different priorities. In the policy evaluation process, only one rule in the set will be fired.

Definition 2 *A reduction R^* w.r.t. P^* is obtained in this way: for each conflicting set w.r.t P^*, keep one rule and delete the others from P^*.*

A reduction is a "flattened" program in the sense that regarding $trust(\boldsymbol{a})$ ($distrust(\boldsymbol{a})$) there is only one rule.

Definition 3 *An interpretation I is a model of a reduction R^* w.r.t. P^* iff for all $A :\!- \phi \in R^*$, $I(A) = I(\phi)$ holds.*

Theorem 1 *Given a reduction R^* w.r.t. P^* there exists a least fixed point of the immediate consequence operator Φ_{R^*} defined as $\Phi_{R^*}(I)(A) = I(\phi)$ for all $A :\!- \phi \in R^*$. The least fixed point is the unique $\preceq_k$ minimal model of R^**

The proof follows directly from [38]. The operator Φ_{R^*} is monotonic regarding the knowledge ordering. Therefore the least fixed point exists.

A reduction may not be adequate in the sense that some trust decisions produced from this reduction may not reliable or informative enough. Therefore we introduce some further constraints.

Definition 4 *A reduction w.r.t. P^* is admissible if the head of each labelled rule in this reduction is mapped to a truth value t in the unique $\preceq_k$ minimal model such that t is an admissible answer w.r.t to the threshold defined for the priority level.*

Even with admissible reductions, we may prefer one to the others. The preference ordering is defined as follows:

Definition 5 *A labelled rules stratification $LR_0^*, ..., LR_m^*$ of P^* is obtained from a stratification of P^* by removing all levels that do not contain labelled rules and removing from the remaining levels all the rules that are not labelled.*

Definition 6 *A mapping to a reduction R^* from LR_i^* ($i \in \{0, ..., m\}$) is defined as a set that contains all the labelled rules in R^* that are also in LR_i^*.*

Definition 7 *Let $LR_0^*, ..., LR_m^*$ be a labelled rules stratification of P^*, R_1^* and R_2^* be two reductions w.r.t. P^*, M_{ij} ($i \in \{0, ..., m\}, j \in \{1, 2\}$) be the mapping to reduction R_j^* from LR_i^*. We define a partial order $\preceq_p$:*

1. *$M_{i1} \preceq_p M_{i2}$ iff for every labelled rule r_1 in M_{i1}, we can find a labelled rule r_2 in M_{i2} with the same head (or its counterpart), such that r_2 has a priority higher than or equal to r_1. $M_{i1} \prec_p M_{i2}$ iff $M_{i1} \preceq_p M_{i2}$ and $M_{i1} \neq M_{i2}$.*
2. *$R_1^* \prec_p R_2^*$ iff we can find $i \in \{0, ..., m\}$ such that $M_{i1} \prec_p M_{i2}$ and for any $j < i$, $M_{j1} = M_{j2}$. $R_1^* \preceq_p R_2^*$ if $R_1^* \prec_p R_2^*$ or $R_1^* = R_2^*$.*

The intuition of this preference ordering is that we prefer labelled rules with higher priority and for labelled rules which depend on other labelled rules, we prefer those based on more preferable rules.

Theorem 2 *The ordering $\preceq_p$ is independence of the choice of labelled rule stratification.*

The proof follows directly from the above definitions.

Definition 8 *An admissible reduction R^* is called a preferred admissible reduction w.r.t. P^* iff for every reduction $R^{*'}$ w.r.t. P^*, $R^{*'} \preceq_p R^*$.*

Theorem 3 *For every P^*, there is a unique preferred admissible reduction.*

To prove Theorem 3, we first prove the following lemma:

Lemma 1 *Given a poset $(AD^*, \preceq_p)$ such that AD^* is the set of all the admissible reductions w.r.t. P^*, for every $R_1^*, R_2^* \in AD^*$, the least upper bound exists.*

Proof. Let $LR_0^*, ..., LR_m^*$ be a labelled rules stratification of P^*. For every $R_1^*, R_2^* \in AD^*$, the least upper bound is LUB_{12} which can be found by iterating from $i = 0$ on a set $S = AD^*$:

1. if S has only one element, stop the iteration.
2. otherwise if $M_{i1} = M_{i2}$ where M_{ij} ($j \in \{1, 2\}$) is the mapping to reduction R_j^* from LR_i^*, delete from S all admissible reductions such that the mappings to the admissible reductions from LR_i^* are not equal to M_{i1}.

3. otherwise if ($M_{i1} \neq M_{i2}$), do the following, for every labelled rule r_1 in M_{i1} and labelled rule r_2 in M_{i2} with the same head (or its counterpart), compare the priorities of r_1 and r_2, put into a set M' the labelled rule with the maximum priority. Delete from S all the admissible reductions such that the mappings to the admissible reductions from LR_i^* are not equal to M'. Then delete from S all the elements except one such that the mapping to it from LR_k^* $i < K \leq m$ contains only default assignment rules.

At the end of the iteration, there will be one and only one element left. This is easy to check. In trivial cases the procedure only take branch 2 and the only element left is $LUB_{12} = R_1^* = R_2^*$. In any non-trivial cases which take branch 3, there must be at least one element in S such that the mapping to the element from LR_i^* equals M'. There will be only one element left by definition. Now we show that LUB_{12}, the only element left in S is indeed the least upper bound.

LUB_{12} is an upper bound of R_1^*, R_2^*. In the trivial case $LUB_{12} = R_1^* = R_2^*$, by definition $R_1^* \preceq_p LUB_{12}, R_2^* \preceq_p LUB_{12}$. In the non-trivial case $R_1^* \neq R_2^*$, when the iteration takes branch 3, it is guaranteed that $R_1^* \preceq_p LUB_{12}, R_2^* \preceq_p LUB_{12}$ because $M_{i1} \preceq_p M'$ and $M_{i2} \preceq_p M'$.

LUB_{12} is the least upper bound of R_1^*, R_2^*, i.e. there is no R_3^* such that R_3^* is an upper bound of R_1^*, R_2^* and $R_3^* \preceq_p LUB_{12}$. In the trivial case $LUB_{12} = R_1^* = R_2^*$, $R_3^* = LUB_{12}$. In the non-trivial case, since there is no other M'' such that $M_{i1} \preceq_p M''$ and $M_{i2} \preceq_p M''$ and $M' \preceq_p M''$, and LUB_{12} contains only default assignment rules, which have the lowest priority, for the labelled rules at higher levels, there is no way to find R_3^* that satisfies the requirements.

Because AD^* is finite, by Lemma 1, $(AD^*, \preceq_p)$ has an unique upper bound, i.e. the preferred admissible reduction.

Theorem 4 *For every P^*, the model M described early in this section corresponds to the unique $\preceq_k$ minimal model of the preferred admissible reduction of P^*.*

Proof. Given P^*, the only difference between P^* and its preferred admissible reduction P^{**} is that for each trust(distrust) atom, P^* contains multiple rules each with a different priority while in the preferred admissible reduction there is only one rule. Given a stratification of P^*, P^{**} can be stratified in the same sequence. The immediate sequence operators are the same. During the evaluation of P^*, whenever a labelled rules is fired, it must also be in the preferred admissible reduction. If a labelled rule r in P^* is fired and the corresponding labelled rule in the preferred admissible reduction has a lower priority, then we can find another admissible reduction R with r and obviously $R \not\preceq_p P^{**}$. It contradicts the definition of preferred admissible reduction. If the corresponding labelled rule in the preferred admissible reduction has a higher priority, then it contradicts Section 6.3 where the rules generate admissible answers with the highest priorities should be fired. The evaluation of P^* is literally the same as the evaluation of P^{**}. Therefore the model generated should be the same as generated using P^{**} under the same immediate sequence operators.

B Top Down Policy Interpreter

The top down policy interpreter is a vanilla-style meta-interpreter [39] which answers a query through a recursive goal reduction procedure. Loosely speaking, the interpreter works in this way: when A is queried and a policy rule such as $A :- B, C, D$ can be found, the evaluation of A is replaced by the evaluation of the policy body B, C, D. The predicates B, C, D are then reduced according to the other policy rules, facts and hypothesis. In the end, the interpreter rewrites the query into a tree where the query is the root node and all the leave nodes are truth values. Then the truth values are combined according to the connectives and back-propagated to the root. An example is shown in 4.

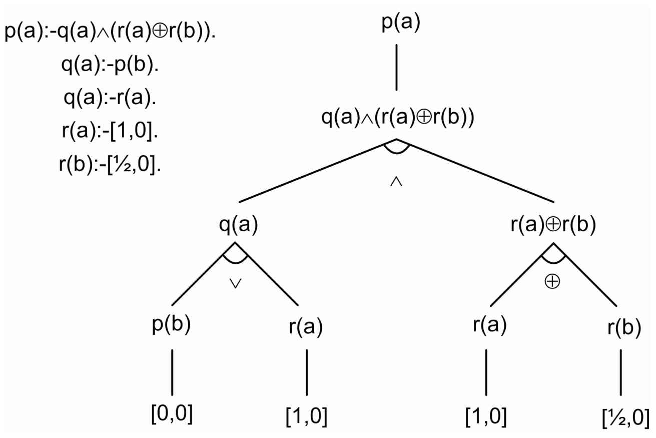

Fig. 4. A rewriting Example

As we can see in the example, the query $p(a)$ is first reduced as $q(a) \wedge (r(a) \oplus r(b))$ according to the policies. Then $q(a)$ is rewritten as $p(b) \vee r(a)$ because there are two rules with head $q(a)$ can be found. $r(a)$ and $r(b)$ are then reduced to their corresponding truth value according to the facts listed and $p(b)$ is reduced to $[0, 0]$ according to the hypothesis. Conceptually, the query can be rewritten as $([0, 0] \vee [1, 0]) \wedge ([1, 0] \oplus [\frac{1}{2}, 0])$ which gives the truth value $[1, 0]$. Although the example shows a ground query and a ground program, the same procedure can be generalised to the cases with variables by simply requiring that unification to be applied as in usual logic programs.

Now let us explain the query evaluation procedure formally. For a query ϕ, an answer is a pair $\langle \theta, b \rangle$ where $\theta = \{\overline{X}/\overline{c}\}$ is a substitution of the variables $\overline{X}$ in ϕ with the constants in $\overline{c}$ and b is a truth value taken from the bilattice truth space. We use $\theta\phi$ to denote the substitution instance of ϕ by applying θ to ϕ. The pseudocode of the query evaluation procedure is shown in Figure 5.

The procedure defines how to interpret and evaluate queries against the policies, facts and hypothesis. We do not consider details of unification and backtracking because they are handled by the underlying Prolog engine. The procedure utilises two

```
procedure: $eval(\phi, \langle\theta, b\rangle)$
input     : query $\phi$
output    : answer $\langle\theta, b\rangle$ where $\theta$ is a substitution and $b$ is a truth value such that the
            intended model $M \models \theta\phi :\!-b$.
if $\phi$ is a truth value then
    $\theta = \{\}$;
    $b = \phi$;
else if $\phi$ is in the form of $\neg\varphi$ then
    $eval(\varphi, \langle\theta, b'\rangle)$;
    $b = \neg b'$;
else if $\phi$ is in the form of $\varphi_1 \circ \varphi_2$, where $\circ$ is one of $\vee, \wedge, \oplus, \otimes$ then
    $eval(\varphi_1, \langle\theta_1, b_1\rangle)$;
    $eval(\theta_1\varphi_2, \langle\theta 2, b_2\rangle)$;
    $b = b_1 \circ b_2$;
    $\theta = \theta 1\theta 2$;
else if $\phi$ is atomic then
    if $\phi$ is an EDB predicate and $\varphi :\!-b'$ is a fact such that $\phi$ unify $\varphi$ with $\theta'$ then
        $\theta = \theta'$;
        $b = b'$;
    else
        $\varphi :\!-b'$ is in hypothesis such that $\phi$ unify $\varphi$ with $\theta'$;
        $\theta = \theta'$;
        $b = b'$;
    end
    if $\phi$ is an IDB predicate then
        if $\phi$ is a trust or distrust predicate then
            $evalTrust(\phi, \langle\theta, b\rangle)$;
        else if $\phi$ is a constraint predicate then
            $evalConstraint(\phi, \langle\theta, b\rangle)$;
        else
            find all policies $p_1, ..., p_n$ such that $\phi$ and $head(p_i)$ unify with $\theta_1$;
            $\varphi = \theta_1(body(p_1)\vee, ... \vee body(p_n))$;
            $eval(\varphi, \langle\theta, b\rangle))$
        end
    end
end
```

Fig. 5. Query evaluation procedure

sub-procedures: $evalTrust$ and $evalConstraint$. $evalTrust$ evaluates trust and distrust goals. $evalConstraint$ evaluates constraints. The pseudocode of these two sub-procedures are listed below (Figure 6, 7):

$evalTrust$ evaluates trust (distrust) policies from the highest priority level. As described in section 6.2, for each level, two thresholds in terms of $\preceq_t$ or $\preceq_k$ or both are defined, one for distrust policies and one for trust policies. Answers that satisfy the threshold are called *admissible answers*. If an admissible answer can be found, the

```
procedure: $evalTrust(\phi, \langle\theta, b\rangle)$
input    : query $\phi$
output   : answer $\langle\theta, b\rangle$.
int level = $maxPriorityLevel$;
bool admissible = $false$;
while admissible is $false$ and level $> 0$ do
    bool renamed =$false$;
    if $\phi$ is a trust predicate then
        change the predicate name of $\phi$ to $distrust$;
        renamed = $true$;
    end
    for all distrust policies $p_1, ..., p_n$ with priority level $level$ such that
    $a = \phi\theta_0, a = head(p_i)\theta_i, (i = 1, .., n)$;
    $\varphi = body(p_1)\theta_1 \vee, ... \vee body(p_n)\theta_n$;
    $eval(\varphi, \langle\theta', b\rangle)$;
    $\theta = \theta_0\theta'$;
    if $b$ is an admissible answer then
        admissible = $true$;
        if renamed is true then
            $b = \neg b$;
        end
    else
        change the predicate name of $\phi$ to $trust$;
        bool renamed = ¬renamed;
        find all trust policies $p_1, ..., p_n$ with priority level $level$ such that
        $a = \phi\theta_0, a = head(p_i)\theta_i, (i = 1, .., n)$;
        $\varphi = body(p_1)\theta_1 \vee, ... \vee body(p_n)\theta_n$;
        $eval(\varphi, \langle\theta', b\rangle)$;
        $\theta = \theta_0\theta'$;
        if $b$ is an admissible answer then
            admissible = $true$;
            if renamed is true then
                $b = \neg b$;
            end
        end
    end
    $level = level - 1$;
end
if admissible is false then
    $\theta = \{\}$;
    $b = [0, 0]$;
end
```

Fig. 6. $evalTrust$ procedure

evaluation ends and the answer is returned. Otherwise the interpreter moves to the next level with lower priority.

```
procedure: $evalConstraint(\phi, \langle\theta, b\rangle)$
input      : query $\phi$
output     : answer $\langle\theta, b\rangle$.
$\phi$ is linked to predicate $\varphi$;
$eval(\varphi, \langle\theta_1, b\rangle)$;
$solve(\theta_1\phi) = (\theta_2, t)$;
$\theta = \theta_2\theta_1$;
if $t$ is false then
    $b = \neg b$;
end
```

Fig. 7. $evalConstraint$ procedure

It is easy to see that under the syntactic restrictions we posed earlier in Section 6.2 (finite Herbrand base, tractable constraint domain, no cyclic dependency between ground atoms), the query evaluation procedure guarantees to terminate. Finite Herbrand base ensures that the evaluation tree is finite, tractable constraint domain ensures that the $evalConstraint$ sub-routine terminates and no cyclic dependency ensures that the reduction process is loop-free.