A Semantic Foundation for Trust Management Languages with Weights: An Application to the *RT* Family*,**

Stefano Bistarelli[1,2], Fabio Martinelli[2] and Francesco Santini[2,4]

[1] Dipartimento di Scienze, Università "G. D'Annunzio" di Chieti-Pescara, Italy
bista@sci.unich.it
[2] Istituto di Informatica e Telematica (CNR), Pisa, Italy
[stefano.bistarelli,fabio.martinelli,francesco.santini@]iit.cnr.it
[3] IMT - Institute for Advanced Studies, Lucca, Italy
f.santini@imtlucca.it

Abstract. In this paper, we present a variant of Datalog language (we call it Datalog^W) able to deal with weights on ground facts and to consequently compute a feedback result for the goal satisfaction. The weights are chosen from a proper c-semiring. In our context, our goal is to use this language as a semantic foundation for languages for expressing trust relationships. As a matter of fact, many of them have a semantics given in terms of crisp constraints: our approach is to extend them to cover also the soft case. Thus, we apply Datalog^W as the basis to give a uniform semantics to declarative RT^W (Trust Management) language family. The approach is rather generic and could be applied to other trust management languages based on Datalog, as a semantic sublayer to represent trust management languages where the trust level is relevant.

1 Introduction and Motivations

Trust is a very interesting and relevant notion in modern pervasive computer systems. It lies at the heart of human interactions and thus as soon as these interactions happen through (and among) digital devices, such trust relationships must be represented, specified, analyzed, negotiated and composed in those systems [11]. As a matter of fact, when one wants to mechanize the reasoning in certain situations, a formalization is necessary. If one wants also to achieve a common understand and comparison among different trust management system, a semantic mechanism would be extremely useful.

To make a concrete example, a Trust Management (TM) language is required to have the expressivity power to represent the trust-related facts of the considered dominion and a method to derive new assessments and decision starting from these base facts. Current trust management languages based on credentials

* The first and third authors are supported by the MIUR PRIN 2005-015491.
** The second author is supported by the EU projects GRIDtrust and SENSORIA.

(for both expressing facts and access policy rules) uses several foundational approaches. However, facts and access rules are not so crisp in the real complex world. For example, each piece of information could have a confidence value associated with it and representing a reliability estimation, or a fuzzy preference level or a cost to be taken in account. The feedback final value, obtained by aggregating all the ground facts together, can be then used to improve the decision support system by basing on this preference level instead of a plain "yes or no" result (e.g, see [15, 6, 5]). In this scenario, a credential could state that the referred entity is a "student" or a "bright student" with a probability of 80% because her/his identity of student is based on what an acquaintance asserts (thus, it is not as certain as declared in IDs), or, in the second case, because the received marks need to be globally evaluated. In literature there are many examples where trust or reputation are computed by aggregating some values together [11], for example in PGP systems, or for generic trust propagation inside social networks. We think that similar quantitative measurements are useful also for trust languages, in order to have a more informative result.

For this reason, we describe a weighted version of Datalog (i.e. Datalog^W) where the rules are enhanced with values taken from a proper c-semiring structure [1, 3], in order to model the preference/cost system; then, we use it as the basis to give declarative semantics to a Role-based Trust-management language according to the principles of RT_0 [14], and called here RT_0^W: the statements of RT_0^W are "soft", i.e. have a related c-semiring value. A similar improvement can be accomplished also for RT_1 [14], i.e. RT_0 extended with parameterized roles. Similar variations for RTML family languages were defined and implemented by using different formal tools in [15]. There, an initial comparison (and integration) between rule-based trust management (RTML) and reputation-based trust systems has been performed and a preliminary (ad-hoc) implementation RTML weighted presented in [8] for GRID systems. However, having a uniform semantics approach to model these languages (as Datalog^W) could be very useful to provide a common understanding as well as a basis for systematic comparison and uniform implementation.

Indeed, there are good reasons to prefer a language that is declarative and has a formal foundation. In this sense, we are following a similar approach as done in [13] for RTML trust management languages, where Datalog with constraints have been proposed as a formal semantics for trust management languages. Since trust is not necessarily crisp, Datalog^W could be used to give formal semantics to this kind of languages with "soft credentials". In this paper we show an approach for RTML that can be further extended to other Datalog-based languages. The main contribution of this paper is thus to provide a formal semantics for such languages that could also bring to a uniform implementation approach, as well as to a comparison among these languages . Giving weights to facts and rules contributes also towards bridging the gap between "rule-based" trust management (i.e. hard security mechanisms) and "reputation based" trust management [11] (i.e. soft security mechanisms).

It is also worth noticing that c-semirings are a valuable mechanism to model and solve optimization problems in several contexts. With our proposal of mix-

ing credential based languages with soft-constraints based on c-semiring in a systematic way, we pave the way for linguistic mechanisms for making optimization decision related to the trust domain. Indeed, this domain could be also coupled with other parameters and thus creating a much more complex (self) optimization mechanisms. For instance, one could use a cost/preference parameter associated with the trust level. The composition of the trust semiring and the preference one is yet amenable of mechanization and this yet leads to a similar treatment we describe here.

In this paper we extend the ideas presented in [2] by giving a weighted semantics to all the *RT* languages presented in [14]. In Sec. 2 we describe the background notions about trust languages and c-semirings. In Sec. 3 we present a weighted version of Datalog, i.e. Datalog^W, while Sec. 4 features the weighted *RT* language family based on Datalog^W, i.e. RT_0^W, RT_1^W, RT_2^W, RT^{WT} and RT^{WD}. At last, in 5 we present the final conclusions.

2 Background

Datalog was originally developed as a query and rule language for deductive databases and is syntactically equivalent to a subset of the Prolog language. Several TM languages are based on Datalog, e.g., Delegation Logic [12], the *RT* (Role-based Trust-management) framework [14], SD3 (Secure Dynamically Distributed Datalog) [10] and Binder [9]. These are some of the languages that can benefit from the semantic basis presented in this paper, even if we will focus only in the *RT* language family.

The *RT* framework is a family of Role-based Trust-management languages [14], whose most basic part is RT_0 which has been then extended to RT_1 with parameterized roles: *University.professorOf(student)* is a statement that can be used to name the professor of a student. An *entity* (or *principal*, e.g. A or B) in *RT* is a uniquely identified individual or process, which can issue credentials and make requests. *RT* assumes that an entity that issued a particular credential or a request can be determined through the use of public/private key pairs. A *role* in *RT* takes the form of an entity followed by a role name (e.g. R with subscripts), separated by a dot. A role defines a set of entities who are members of this role: each entity A has the authority to define who are the members of each role of the form $A.R$. Each statement defines one role to contain either an entity, another role, or certain other expressions that evaluate to a set of entities. More details will be given in Sec. 4.

An important extension that significantly enhances the expressivity of this kind of languages is presented in [13]. In that work, the authors present Datalog extended with constraints (denoted by Datalog$^{\mathcal{C}}$) in order to define access permissions over structured resources as trees.

Several approaches advocated the usage of trust levels w.r.t. attributes, also stated directly in digital credentials. In addition to the works on the extension of RTML with weights and its relationships with other trust models as the Josang one already mentioned [15, 8], there is also the work on policy and reputation done in [5]. Here the PROTUNE policy language is extended to deal with trust

and reputation levels. Also role based access control has been extended with trust levels in [6]. All these works use specific logics and approaches.

C-semirings A c-semiring S [1, 3] (or simply semiring in the following) is a tuple $\langle S, +, \times, \mathbf{0}, \mathbf{1}\rangle$ where S is a set with two special elements ($\mathbf{0}, \mathbf{1} \in S$) and with two operations $+$ and $\times$ that satisfy certain properties: $+$ is defined over (possibly infinite) sets of elements of S and thus is commutative, associative, idempotent, it is closed and $\mathbf{0}$ is its unit element and $\mathbf{1}$ is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, $\mathbf{1}$ is its unit element, and $\mathbf{0}$ is its absorbing element (for the exhaustive definition, please refer to [1, 3]). The $+$ operation defines a partial order $\leq_S$ over S such that $a \leq_S b$ iff $a + b = b$; we say that $a \leq_S b$ if b represents a value *better* than a. Notice that the partial order can be defined since the $+$ operator is commutative, associative and idempotent. Other properties related to the two operations are that $+$ and $\times$ are monotone on $\leq_S$, $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum, $\langle S, \leq_S\rangle$ is a complete lattice and $+$ is its lub. Finally, if $\times$ is idempotent, then $+$ distributes over $\times$, $\langle S, \leq_S\rangle$ is a complete distributive lattice and $\times$ its glb.

Varying the set S and the meaning of the $+$ and $\times$ operations, we can represent many different kinds of problems, having features like fuzziness, probability, and optimization. Moreover, in [3] the authors have shown that the cartesian product of two c-semirings is another c-semiring, and this can be fruitfully used to describe multi-criteria constraint satisfaction and optimization problems, e.g. the *path semiring* presented in Sec. 3.

3 A Weighted Extension of Datalog

Datalog is a restricted form of logic programming with variables, predicates, and constants, but without function symbols. Facts and rules are represented as Horn clauses in the generic form R_0 :- $R_1, \ldots, R_n$. A Datalog rule has the form $R_0(t_{0,1}, \ldots, t_{0,k_0})$: -$R_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, R_n(t_{n,1}, \ldots, t_{n,k_n})$, where $R_0, \ldots, R_n$ are predicate (relation) symbols and each term $t_{i,j}$ is either a constant or a variable ($0 \leq i \leq n$ and $1 \leq j \leq k_i$). The formula $R_0(t_{0,1}, \ldots, t_{0,k_0})$ is called the head of the rule and the sequence $R_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, R_n(t_{n,1}, \ldots, t_{n,k_n})$ the body. If $n = 0$, then the body is empty and the rule is called a fact. Moreover, each program P in Datalog (i.e. a finite set of rules) must satisfy two *safety* conditions: *i)* all variables occurring in the head of a rule also have to appear in the body, and *ii)* every fact in P must be a ground fact.

We can now define our *Weighted Datalog*, or Datalog^W based on classical Datalog. While rules have the same form as in classical Datalog, a fact in Datalog^W has the form: $R_i(x_{i,1}, \ldots, x_{i,k_i})$: - s. Therefore, the extension is obtained by associating to ground facts a value $s \in S$ taken from the semiring $\langle S, +, \times, \mathbf{0}, \mathbf{1}\rangle$. This value describes some properties of the fact, depending on the chosen semiring: for example, we can add together all these values by using the *Weighted* semiring $\langle R^+, min, +, \infty, 0\rangle$, trying to minimize the overall sum at the same time. Otherwise, we can find the best global preference level by using the *Fuzzy*

semiring $\langle [0,1], max, min, 0, 1\rangle$ or we can retrieve the highest resulting probability when we compose all the ground facts, by using the *Probability* semiring $\langle [0,1], max, \times, 0, 1\rangle$.

s(X) :- p(X,Y).	q(a) :- t(a).
p(a,b) :- q(a).	t(a) :- 2.
p(a,c) :- r(a).	r(a) :- 3.

Table 1. A simple Datalog^W program.

Table 1 shows an example of Datalog^W program, for which we suppose to use the *Weighted* semiring. The intuitive meaning of a semiring value like 3 associated to the atom $r(a)$ (in Tab. 1) is that $r(a)$ costs 3 units. Thus the set N contains all possible costs, and the choice of the two operations min and $+$ implies that we intend to minimize the sum of the costs. This gives us the possibility to select the atom instantiation which gives the minimum cost overall. Given a goal like $s(x)$ to this program, the operational semantics collects both a substitution for x (in this case, $x = a$) and also a semiring value (in this case, 2) which represents the minimum cost among the costs for all derivations for $s(x)$. To find one of these solutions, it starts from the goal and uses the clauses as usual in logic programming, except that at each step two items are accumulated and combined with the current state: a substitution and a semiring value (both provided by the used clause). The combination of these two items with what is contained in the current goal is done via the usual combination of substitutions (for the substitution part) and via the multiplicative operation of the semiring (for the semiring value part), which in this example is the arithmetic $+$. Thus, in the example of goal $s(X)$, we get two possible solutions, both with substitution $X = a$ but with two different semiring values: 2 and 3. Then, the combination of such two solutions via the min operation give us the semiring value 2.

To compute trust, in Sec. 4.1 we will use the *path semiring* [16]: $S_{trust} = \langle\langle [0,1], [0,1]\rangle, +_p, \times_p, \langle 0,0\rangle, \langle 1,1\rangle\rangle$, where

$$\langle t_i, c_i\rangle +_p \langle t_j, c_j\rangle = \begin{cases} \langle t_i, c_i\rangle & \text{if } c_i > c_j, \\ \langle t_j, c_j\rangle & \text{if } c_i < c_j, \\ \langle max(t_i, t_j), c_i\rangle & \text{if } c_i = c_j. \end{cases}$$

$$\langle t_i, c_i\rangle \times_p \langle t_j, c_j\rangle = \langle t_i t_j, c_i c_j\rangle$$

In this case, trust information is represented by a couple of values $\langle t, c\rangle$: the second component represents a trust value in the range $[0,1]$, while the first component represents the accuracy of the trust value assignment (i.e. a *confidence* value), and it is still in the range $[0,1]$. This parameter can be assumed as a *quality* of the opinion represented instead by the trust value; for example, a high confidence could mean that the trustor has interacted with the target for a long time and then the correlated trust value is estimated with precision.

Finite Computation Time Being the Datalog^W language a subset of the *Soft Constraint Logic Programming* language [4] with no functions, we can can use the results in [4] to prove that, considering a fixed Datalog^W program, the time for computing the value of any goal for this program is finite and bounded by a constant. The reason is that we just have to consider a finite subclass of refutations (i.e. *simple refutations*) with a bounded length. After having considered all these refutations up to that bounded length, we have finished computing the semiring value of the given goal. Given a refutation tree, a path from the root to a leaf is called *simple* if all its nodes have different labels up to variable renaming. A refutation is a simple refutation if all paths from the root to a leaf in its refutation tree are simple. The proof of Theo. 1 is given in [4].

Theorem 1 (Finite Set of Simple Refutations). *Given a Datalog^W program P and a goal C, consider the set $SR(C)$ of simple refutations starting from C and building the empty substitution. Then $SR(C)$ is finite.*

4 Extending the *RT* Family with Datalog^W

We describe four kinds of credentials for defining roles in a TM language family, here called RT^W, which is based on Datalog^W (see Sec. 3). This family uniformly extends the classical RT family [14] by associating a weight, or better, a semiring value to the basic role definition. Therefore, all the following credentials must be parameterized with a chosen $\langle S, +, \times, \mathbf{0}, \mathbf{1}\rangle$ semiring in order represent preference/cost or fuzzy information associated to the statements. For every following RT_0^W credential, we describe how it can be translated in a corresponding Datalog^W rule. Then we will suggest how to extend RT_0^W with parameterized roles, obtaining the RT_1^W language.

Rule 1 $A.R \longleftarrow \langle B, s\rangle$ where A and B are (possibly the same) entities, and R is a role name. This means that A defines B to be a member of A's R role. This statement can be translated to Datalog^W with the rule $r(A, B)$:- s, where s is the semiring value associated with the related ground fact, i.e. $s \in S$.

Rule 2 $A.R \longleftarrow B.R_1$ This statement means that A defines its R role to include (all members of) B's R_1 role. The corresponding Datalog^W rule is $r(A, x)$:- $r_1(B, x)$.

Rule 3 $A.R \longleftarrow A.R_1.R_2$, where $A.R_1.R_2$ is defined as *linked role* [14] and it means that A defines its R role to include (the members of) every role $B.R_2$ in which B is a member of $A.R_1$ role. The mapping to Datalog^W is $r(A, x)$:-$r_1(A, y), r_2(y, x)$.

Rule 4 $A.R \longleftarrow B_1.R_1 \cap B_2.R_2 \cap \cdots \cap B_n.R_n$. In this way, A defines its R role to include the intersection of the n roles. It can be translated to Datalog^W with $r(A, x)$:-$r_1(B_1, x), r_2(B_2, x), \ldots, r_n(B_n, x)$.

The semantics of a program using these rules will find the best credential chain according to the + operator of the chosen semiring, which defines a partial order $\leq_S$. Notice that only the basic role definition statement (i.e. **Rule 1**) is

enhanced with the semiring value $s \in S$, since the other three rules are used to include one role into another or to obtain the intersection of different roles.

Notice that having a semiring value associated only with ground facts does not prevent us from giving a weight also to rules. This can be accomplished by slightly changing the syntax of the credentials used to compose the roles together (i.e. **Rules 1-2-3**), by associating a semiring value also to them. Then, in the Datalog translation, a new ground fact can be added in the body of the rule, whose weight models the use of that specific rule. For example, **Rule 2** becomes $A.R \longleftarrow \langle B.R_1, s\rangle$ (where s is a value taken from the same S semiring set), and its Datalog translation is $r(A,x)$:-$r_1(B,x), rule\_weight$, where $rule\_weight$:- s is the ground fact that gives a weight to the rule. Clearly, nothing changes from the computational point of view (see Sec 3).

It is easy to extend this language in order to enhance it with parameterized roles, thus obtaining a RT_1^W language following the hierarchy presented in [14]. This parametrization can be used to represent relationships among entities, e.g. *University.professorOf(student)* to name the professor of a student, but also to represent attributes that have fields, e.g. the number of exams or the enrollment academic year and so on. With respect to the previous four rules, in RT_1^W the *head* of a credential has the form $A.R(h_1,\ldots,h_n)$, in which A is an entity, and $R(h_1,\ldots,h_n)$ is a role name (R is a role identifier). For each $i \in 1\ldots n$, h_i is a data term having the type of the ith parameter of R. For example, **Rule 1** can be rewritten in RT_1^W as $A.R(h_1,\ldots,h_n) \longleftarrow \langle B, s\rangle$, and mapped to Datalog^W as $r(A,B,h_1,\ldots,h_n)$:- s. Our intention is to extend the RT^W family according to the guidelines explained in [14] (see Sec. 5).

Since Datalog is a subset of first-order logic, the semantics of a TM language based on it is declarative and unambiguous. While The $\times$ operator of the semiring is used to compose the preference/cost values associated to the statements, the $+$ is used to let the framework select the best derivation with more chances to authorize the requester (among all the credentials revealed by her/him).

In the next theorem we claim that our weighted language family can be used to represent also classical RT credentials [14]. In this sense, the RT^W languages can be considered as a foundation layer for all the classical RT languages (RT_2^W will be instead presented in Sec. 4.2).

Theorem 2 (Language Family Inclusion). *For each S set of statements in the RT_0, RT_1 or RT_2 language, we can find a corresponding S^W set of statements respectively represented in RT_0^W, RT_1^W or RT_2^W, and whose semantics is the same. This can be accomplished by using Datalog^W together with the* Boolean *semiring.*

In Fig. 1 we show the result of Theo. 2, i.e. the vertical inclusions; the horizontal ones are explained in [14] (for RT) and in this paper (for RT^W). Theorem 2 can be proved by using the *Boolean* semiring $\langle\{0,1\},\vee,\wedge,0,1\rangle$ and by assigning a weight of 1 (i.e. the *true* value) to all the ground facts. In this way we obtain a set of crisp statements and the semantics returns all the possible derivations, as the corresponding RT set of statements would do.

In Sec. 4.3 and Sec. 4.4 we respectively introduce other two RT-based languages: RT^{WT} and RT^{WD} can be used, together or separately, with each of RT_0^W, RT_1^W, or RT_2^W. The resulting combinations are written as RT_i^W, RT_i^{WT} and RT_i^{WD} for $i = 0, 1, 2$.

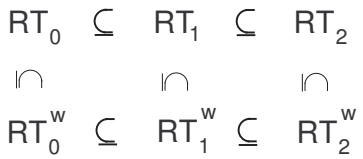

Fig. 1. A hierarchy of RT^W languages, compared with the classical RT one.

4.1 Some Examples with Levels of Trust

We can start by adding levels to the classical RT_0 example presented in many RT related papers (e.g. [14]). To solve the example in Tab. 2, we use a *Fuzzy* semiring $\langle [0,1], max, min, 0, 1 \rangle$, where the elements in $[0,1]$ represents the truth degree connected to a credential and evaluated by the entity which signs and issues it: for example, `StateU.highMarks` $\longleftarrow$ $\langle$ `Alice, 0.8` $\rangle$ in Tab. 2 certifies that Alice has obtained a good number of high marks (since the value is 0.8) for the exams completed at the StateU university (the credential is issued by StateU).

```
EPub.disct ⟵ EPub.preferred ∩ EPub.brightStudent.
EPub.preferred ⟵ EOrg.highBudget ∩ EOrg.oldCustomer.
EPub.brightStudent ⟵ EPub.goodUniversity.highMarks.
EPub.goodUniversity ⟵ ABU.accredited.
ABU.accredited ⟵ ⟨ StateU, 0.9 ⟩.
StateU.highMarks ⟵ ⟨ Alice, 0.8 ⟩.
EOrg.highBudget ⟵ ⟨ Alice, 0.6 ⟩.
EOrg.oldCustomer ⟵ ⟨ Alice, 0.7 ⟩.
```

Table 2. An example in RT_0^W, with fuzzy values associated to the credentials.

The example in Tab. 2 describes a fictitious Web publishing service, *EPub*, which offers a discount to anyone who is both a preferred customer and a bright student. EPub delegates the authority over the identification of preferred customers to its parent organization, *EOrg*. In order to be evaluated as a preferred customer, EOrg must issues two different types of credentials stating that the customer is not new (i.e. *EOrg.oldCustomer*) and has already spent some money in the past (i.e. *EOrg.highBudget*). EOrg assigns a fuzzy value to both these

two credentials to quantify its evaluation. EPub delegates the authority over the identification of bright students to the entities that are accredited universities. To identify such universities, EPub accepts accrediting credentials issued by the fictitious *Accrediting Board for Universities* (ABU). ABU evaluates a university with a fuzzy score and each university evaluates its enrolled students. A student is bright if she/he is both enrolled in a good university and has high marks. The final fuzzy score, obtained by composing together all the values of the used credentials, can be compared with a threshold to authorize the discount: e.g. only entities whose set of credentials produced a score greater than 0.7 are authorized. Otherwise, the final fuzzy result can be used to derive a proportional discount amount: for example a score of 0.8 could authorize a discount that is twice the discount allowed with a score of 0.4. The following credentials prove that Alice is eligible for the discount with a score of 0.6, determined by the fact that she has not a very high budget spent at EOrg (i.e. her $EOrg.highBudget$ credential has a value of 0.6).

$$
\begin{aligned}
\texttt{EPub.disct} &\longleftarrow \texttt{EPub.preferred} \cap \texttt{EPub.brightStudent}.\\
\texttt{EPub.disct} &\longleftarrow \texttt{EOrg.famousProf.goodRecLetter}.\\
\texttt{EPub.preferred} &\longleftarrow \texttt{EOrg.highBudget} \cap \texttt{EOrg.oldCustomer}.\\
\texttt{EPub.brightStudent} &\longleftarrow \texttt{EPub.goodUniversity.highMarks}.\\
\texttt{EPub.goodUniversity} &\longleftarrow \texttt{ABU.accredited}.\\
\texttt{EOrg.famousProf} &\longleftarrow \langle\ \texttt{ProfX},\ \langle\ 0.9,\ 0.9\ \rangle\rangle.\\
\texttt{ProfX.goodRecLetter} &\longleftarrow \langle \texttt{Alice},\ \langle\ 0.9,\ 0.8\ \rangle\rangle.\\
\texttt{ABU.accredited} &\longleftarrow \langle\ \texttt{StateU},\ \langle\ 0.9,\ 0.8\ \rangle\rangle.\\
\texttt{StateU.highMarks} &\longleftarrow \langle\ \texttt{Alice},\ \langle\ 0.8,\ 0.9\ \rangle\rangle.\\
\texttt{EOrg.highBudget} &\longleftarrow \langle\ \texttt{Alice},\ \langle\ 0.6,\ 0.5\ \rangle\rangle.\\
\texttt{EOrg.oldCustomer} &\longleftarrow \langle\ \texttt{Alice},\ \langle\ 0.7,\ 0.7\rangle\rangle.
\end{aligned}
$$

Table 3. An extension of the example in Tab. 2, using the *path semiring*.

In Tab. 3 we extend the example of Tab. 2 in order to represent also a case where the authorization can be accomplished by following different derivations. For example, a customer could be allowed to have a discount even if she/he presents a good recommendation letter written by a famous professor (i.e. *EPub.famousProf.goodRecLetter*). In Tab. 3 we use the *path semiring* presented in Sec. 3, thus a semiring value consists in a couple of trust/confidence feedbacks. The best derivation corresponds to the criteria defined by the $+_p$ (i.e. confidence is more important).

4.2 RT_2^W: Logical Rights

Trust languages can be used to grant some permissions, i.e. to represent access modes over some specific objects. For this reason it useful to group logically related objects (e.g. the files inside the same directory) and access modes, and to give permissions about them in a correlated manner. As proposed in [14], we

introduce in our language the notion of *o-sets*, which are used to group together this kind of objects: o-sets names are created by associating an o-set identifier to a tuple of data terms. Moreover, an o-set identifier has a base type τ, and o-set names/o-sets created by using an o-set identifier have the same base type as the o-set identifier. Finally, the value of an o-set is a set of values in τ.

An o-set-definition credential is similar to the role definition credential that we have defined in Sec. 4 for RT_1^W: the difference is that the members of o-sets are objects that are not entities. $\texttt{Admin.Documents}(\texttt{read}) \longleftarrow \langle \texttt{FileA}, 0.9 \rangle$, for example, states that the administration office grants to *FileA* the permission to be read only for the 90% of it; *FileA* and the *Documents* o-set id are associated with the *file* type.

O-set-definition credentials can be translated in Datalog exactly as proposed for RT_1^W in Sec. 4: the *head* of a credential has the form $A.O(h_1, \ldots, h_n)$, where $O(h_1, \ldots, h_n)$ is an o-set name of type τ, while the body can be a value of base type τ, another o-set $B.O_1(s_1, \ldots, s_m)$ of base type τ, a linked o-set $A.R_1(t_1, \ldots, t_l).O_1(s_1, \ldots, s_m)$, in which $R_1(t_1, ..., t_l)$ is a role name and $O_1(s_1, \ldots, s_m)$ is an o-set name of base type τ, or an intersection of k o-sets of the base type τ (see Sec. 4.3 for the intersection of roles and o-sets).

Therefore, a credential in RT_2^W is either a role-definition credential or an o-set-definition credential. For more details on types and properties of RT_2^W credentials (w.r.t. RT_1^W), please refer to [14].

Example 1. In this example, the *AlphaCompany* allows the members of a project team to work on the documents of this project: each of the credentials representing the documents, e.g. a $fileA$ file, are associated with a couple of values, e.g. $\langle 0.9, 0.5 \rangle$, which grant a member of the project the right to read 90% of the file and to modify 50% of it. This restriction on files can be explained by copyright or *Concurrent Versioning System* limitations, or due to the different position taken by employees. Even the credentials concerning the members of the project (e.g. *Bob*) are weighted with the same percentages, in this case related instead to the role of the entity (i.e. there are generic read/modify rights associated to *Bob*): in this way, it possible to combine all these levels of rights together, and to finally know how much a given entity can read and modify a given object. As we presented in Sec. 2, the cartesian product of two c-semirings is still a c-semiring and, therefore, it is not a problem to have multiple weights (more details are given in [3]); for this reason we use the vectorization of two *Fuzzy* semirings, i.e. $\langle \langle [0,1], [0,1] \rangle, \langle max, max \rangle, \langle min, min \rangle, \langle 0, 0 \rangle, \langle 1, 1 \rangle \rangle$, in order to maximize (i.e. with $\langle max, max \rangle$) the composition of the values representing the rights, (i.e. with $\langle min, min \rangle$): in practice, we use the *Fuzzy* semiring to find the maximum read/modify percentages, obtained by keeping the worst value among all the composition percentages. The credentials to represent this scenario are the following ones, from which we can obtain $\texttt{AlphaCompany.fileAc}(\texttt{read}, \texttt{modify}, \texttt{fileA}) \longleftarrow \texttt{Bob}$ with a value of $\langle 0.8, 0.5 \rangle$:

$$\texttt{AlphaCompany.fileAc}(\texttt{read}, \texttt{modify}, \texttt{AlphaCompany.documents}(\texttt{x})) \longleftarrow \texttt{AlphaCompany.team}(\texttt{x}).$$

$$\texttt{AlphaCompany.documents(proj)} \longleftarrow \langle \texttt{fileA}, \langle 0.9, 0.5 \rangle \rangle.$$
$$\texttt{AlphaCompany.team(proj1)} \longleftarrow \langle \texttt{Bob}, \langle 0.8, 0.7 \rangle \rangle.$$

4.3 RT^{WT}: Threshold and Separation-of-Duty Policies

Threshold structures are satisfied by the agreement of k out of a set of entities that satisfy a specified condition, while *separation of duty* instead requires that two or more different people be responsible for the completion of a sensitive task, such deciding the result of an exam. With **Rule 4** (see Sec. 4) it is possible to implement simple threshold structures by using the intersection of roles; for example, the policy stating that a student is considered bright (bS) by her/his university (Uni) only if two out of three professors ($P1$, $P2$ and $P3$) say so, can be represented by the three rules $Uni.bS \longleftarrow P1.bS \cap P2.bS$, $Uni.bS \longleftarrow P1.bS \cap P3.bS$ and $Uni.bS \longleftarrow P2.bS \cap P3.bS$.

However, with this kind of intersections we are not able express complex policies: for example if we need to represent the fact that A says that an entity has attribute R if two different entities having attribute R_1 says so. For this reason we need to introduce the RT^{WT} language, in order to properly work with sets of entities. More specifically, RT^{WT} adds to the RT^{W} languages the notion of *manifold roles*, which generalizes the notion of roles [14]. A manifold role has a value that is a set of entity collections. An entity collection is either an entity, which can be viewed as a singleton set, or a set of two or more entities. Notice that, as the RT^{WD} language presented in Sec. 4.4, RT^{WT} can be used together with each of RT_i^W languages (see Sec. 4). In RT^{WT} we introduce two more types of credentials w.r.t. Sec. 4:

Rule 5 $A.R \longleftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$. As we introduced before with words, the meaning of this credential is $members(A.R) \supseteq members(B_1.R_1 \odot \cdots \odot B_k.R_k) = \{s_1 \cup \cdots \cup s_k | s_i \in members(B_i.R_i) \text{ for } 1 \leq i \leq k\}$. Given $w_1, \ldots w_k$ as the actual weights of the derivations respectively rooted in $B_1.R_1, \ldots B_k.R_k$, the global weight of this clause is then composed as $w_1 \times w_2 \times \cdots \times w_k$, where $\times$ depends on the chosen $\langle S, +, \times, \mathbf{0}, \mathbf{1} \rangle$ semiring.

Rule 6 $A.R \longleftarrow B_1.R_1 \otimes \cdots \otimes B_k.R_k$. The formal meaning of this credential is instead given by $members(A.R) \supseteq members(B_1.R_1 \otimes \cdots \otimes B_k.R_k) = \{s_1 \cup \cdots \cup s_k | (s_i \in members(B_i.R_i) \wedge s_i \cap s_j = \emptyset) \text{ for } 1 \leq i \neq j \leq k\}$. Given $w_1, \ldots w_k$ as the actual weights of the derivations respectively rooted in $B_1.R_1, \ldots B_k.R_k$, the global weight of this clause is then composed as $w_1 \times w_2 \times \cdots \times w_k$, where $\times$ operator depends on the chosen $\langle S, +, \times, \mathbf{0}, \mathbf{1} \rangle$ semiring.

As usual, the Datalog engine will select the best derivation according to the $+$ operator of the semiring. Considering RT^{WT}, the translation to Datalog rules for **Rule 1**, **Rule 2** and **Rule 4** is the same as the one presented in Sec. 3. For **Rule 3**, **Rule 5** and **Rule 6** rules, the translation is instead the following one:

Rule 3 $A.R \longleftarrow A.R_1.R_2$ can be translated to $r(A, x)$:-$r_1(A, y), r_2(y, x)$ when $size(r_1) = 1$, or can be translated to $r(A, y)$:-$r_1(A, x), r_2(y, x_1), \ldots r_2(y, x_k)$,

$set_k(x, x_1, \ldots, x_k)$ when $size(r_1) = k > 1$. Each role identifier has no a *size*: the size of a role limits the maximum size of each of its member entity set (see [14] for further details). The new set_k predicate takes $k+1$ entity collections as arguments, and $set_k(s, s_1, \ldots, s_k)$ is true if and only if $s = s_1 \cup \cdots \cup s_k$; if s_i is an entity, it is treated as a single-set element.

Rule 5 $A.R \longleftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$ can be translated to $r(A, x)$:-$r_1(B_1, x_1)$, $r_2(B_2, x_2), \ldots, r_k(B_k, x_k), set_k(x, x_1, \ldots x_k)$.

Rule 6 $A.R \longleftarrow B_1.R_1 \otimes \cdots \otimes B_k.R_k$ can be translated to $r(A, x)$:-$r_1(B_1, x_1)$, $r_2(B_2, x_2), \ldots, r_k(B_k, x_k), nset_k(x, x_1, \ldots x_k)$. The $nset_k$ predicate takes $k+1$ entity collections as arguments and it is true only when $s = s_1 \cup \cdots \cup s_k$ and for any $1 \leq i \neq j \leq k, s_i \cap s_j = \emptyset$

Example 2. Suppose that for a university office a student is "bright" (i.e. $Uni.bS$) if one member of $Uni.evalExtAdvisor$ (i.e. an external advisor) and two different members of $Uni.EvalProf$ (i.e. a professor who teaches in that university) all say so. This can be represented using the following credentials (where A, B, C and D can be external advisors and/or professors):

$$\texttt{Uni.bS} \longleftarrow \texttt{Uni.evaluators.bS}.$$
$$\texttt{Uni.evaluators} \longleftarrow \texttt{Uni.evalProfs} \odot \texttt{Uni.evalExtAdvisor}.$$
$$\texttt{Uni.evalProfs} \longleftarrow \texttt{Uni.evalProf} \otimes \texttt{Uni.evalProf}.$$
$$\texttt{Uni.evalExtAdvisor} \longleftarrow \langle \texttt{A}, 0.9 \rangle. \quad \texttt{Uni.evalExtAdvisor} \longleftarrow \langle \texttt{B}, 0.7 \rangle.$$
$$\texttt{Uni.evalProf} \longleftarrow \langle \texttt{A}, 0.8 \rangle. \quad \texttt{Uni.evalProf} \longleftarrow \langle \texttt{C}, 0.8 \rangle.$$
$$\texttt{Uni.evalProf} \longleftarrow \langle \texttt{D}, 0.6 \rangle.$$

If we adopt the *Fuzzy* semiring $\langle [0, 1], max, min, 0, 1 \rangle$, the best authorization corresponds to the set $\{A, C\}$ with a value of 0.8 (i.e. the *min* between 0.9 and 0.8): we remind that A is both a professor (i.e. A teaches at the university) and an external advisor (i.e. A can be a visiting professor) and therefore only two entities can satisfy the request. Therefore, with this program we retrieve the best combination of evaluators for a student: the student is supposed to present her/his signed credentials and to request how much she/he is considered bright: the evaluations of different evaluators are composed by selecting the worst score (with the *min* operation of the semiring), but at the end the best derivation is selected (by using the *max* operator).

4.4 RT^{WD}: Delegation of Role Activations

The RT^{WD} language is finally added to our weighted family in order to handle delegation of the capacity to exercise role memberships. The motivations are that, in many scenarios, an entity prefers not to exercise all his rights. For example, a professor could want to log as a simple university employee, thus not having the rights to insert or change the student exam results, but only having the rights to check the number of canteen tickets. With a weighted extension (i.e. RT^{WD}) we are now able to state "how much" the rights are delegated to another entity (e.g. a session or a process). Therefore it is possible to quantify the "amount" of delegated rights, e.g. to modify a document, but only for the

80% of it, which is for example less than the rights held by the delegating entity (e.g. 100%). The delegation takes the following form: $B_1 \xrightarrow{D\ as\ A.R} B_2$, which means that B_1 delegates to B_2 the ability to act on behalf of D in D's capacity as a member of $A.R$.

For the definition of the RT^{WD} rules we introduce the *forRole* predicate as in [14]: $forRole(B, D, A.R)$ can be read as B *is acting for "D as* $A.R$*"* and it means that B is acting for the role activation in which D activates $A.R$. The delegation rules can be translated in the following way: $B_1 \xrightarrow{D\ as\ A.R} B_2$ $forRole(B_2, D, A.R) \longleftarrow forRole(B_1, D, A.R)$. This rule means that B_2 is acting for "D as $A.R$" if B_1 is doing so. Other kind of delegation rules that can be formulated are presented in [14].

Clearly, even the other rules presented in Sec. 4 and Sec. 4.3 must be modified according to the introduction of the *forRole* predicate. For example, $A.R \longleftarrow \langle D, s\rangle$ becomes $forRole(D, D, A.R)$:- s and $s \in S$ is the associated weight taken from the $\langle S, +, \times, \mathbf{0}, \mathbf{1}\rangle$ semiring. Therefore we have presented only the **Rule 1** translation and, for sake of brevity, we omit all the other rules translation with the *forRole* predicate (from **Rule 2** to **Rule 6**); however the translation is similar to the one proposed in the RT^D design in [14]. A request is translated in the same way as a delegation credential; the request is replaced by the dummy entity corresponding to it. For example, the $B_1 \xrightarrow{D\ as\ A.R} req$ request is translated to $forRole(ReqID,\ D,\ A.R) \longleftarrow forRole(B_1,\ D,\ A.R)$, where $ReqID$ is the dummy entity for req.

Example 3. In this simple example we show how different delegation acts can lead to different costs. We use the *Weighted* semiring, i.e. $\langle R^+, min, +, \infty, 0\rangle$, since we suppose the authorizer wants to minimize the cost associated with the credentials used for the authorization (the + of the semiring is instantiated to *min* in the *Weighted* semiring): the costs are elements of R^+ (i.e. the set of positive real numbers) and are composed with the arithmetic + (i.e. the $\times$ of the *Weighted* semiring). The total cost value can be considered, for example, as the cost charged to the authorizer in order to satisfy the requester. For example, the authorizer is represented by a university budget office, and the cost associated with the credentials represents the money cost to manage them (i.e. phone calls, faxes, travel expenses, etc). In the example, we have a university (i.e. Uni), where any conference organization event has to be proposed and approved before it is allowed to be practically organized. Any professor can propose such an event. A member of the "approval commission" can instead approve an event. A member of this commission is also a professor (i.e. the commission is made up of professors); however, a professor cannot approve his own proposed event. Therefore, the aim of the university budget office is to minimize the cost for the organization of the events. This can be represented as follows:

$$\texttt{Uni.organizeEvent} \longleftarrow \texttt{Uni.propose} \otimes \texttt{Uni.approve}.$$
$$\texttt{Uni.propose} \longleftarrow \texttt{Uni.prof}.$$
$$\texttt{Uni.approve} \longleftarrow \texttt{Uni.appCommission}.$$

$$\texttt{Uni.prof} \longleftarrow \texttt{Uni.appCommission}.$$

Suppose also that A and B professors are both in the approval commission and the cost of these two credentials is the same (e.g. 1 euro is a basic cost to manage a member of the approval commission): $\texttt{Uni.appCommission} \longleftarrow \langle \texttt{A}, 1 \rangle$ and $\texttt{Uni.appCommission} \longleftarrow \langle \texttt{B}, 1 \rangle$.

Both of them wish to propose and clearly accept the same event (named *bigConf*) and they present the following credentials. Moreover, we extend the syntax of the delegation rules as already explained in Sec 4: now they can have an associated semiring value (the cost) taken from R^+.

$$\begin{aligned}
\texttt{A} &\xrightarrow{\texttt{A as Uni.appCommission}} \langle \texttt{event(bigConf)}, 6 \rangle. \\
\texttt{A} &\xrightarrow{\texttt{A as Uni.prof}} \langle \texttt{event(bigConf)}, 5 \rangle. \\
\texttt{B} &\xrightarrow{\texttt{B as Uni.appCommission}} \langle \texttt{event(bigConf)}, 8 \rangle. \\
\texttt{B} &\xrightarrow{\texttt{B as Uni.prof}} \langle \texttt{event(bigConf)}, 2 \rangle.
\end{aligned}$$

Given the request $forRole(reqID,\ \{A, B\},\ Uni.organizeEvent)$ (and $reqID$ is the dummy entity), the system will choose B as the proposer (with a cost of 2) and A as the entity who approves the event (with a cost of 6), since it is the cheapest solution to the problem. The total cost of all the credentials is 10 euro, obtained by summing also 1 euro for each credential related to a professor. Notice that the other possible solution, with A proposer and B approver of the event, costs 15 euro, i.e. 5 euro more.

5 Conclusions and Future Work

We have proposed a weighted extension of Datalog (i.e. Datalog^W) and a trust language family based on it. These languages can be used to deal with vague and imprecise security policies or credentials, and preference or costs associated to each rule or fact. In practice, we can manage and combine together different levels of truth, preference or costs associated to the statements and finally have a single feedback value on which to authorize a trust request. We have extended the RT family [14] and we we have shown that the classical RT_0 and RT_1 languages are respectively included in our RT_0^W and RT_1^W languages. It is worthy to notice that our extension is completely orthogonal w.r.t. the RT extension proposed in [13], i.e. RT^C, where the supporting Datalog^C language allows first-order formulas in tractable constraint domains. The constraints are introduced to represent the access permissions over structured resources, e.g., tree domains and range domains. Our aim is instead the representation of trust levels modelling cost/preference or fuzziness of credentials. Our systematic approach to give weights to facts and rules, contributes also towards bridging the gap between "rule-based" trust management (i.e. hard security mechanisms) and "reputation based" trust management [11] (i.e. soft security mechanisms).

On future improvement could be to leave to the programmer the opportunity to take more decisions inside the rules, for example based on the current aggregated semiring value (the process is called *reification* of the values, i.e.

make them visible to the programmer); from its evaluation, some rules could be enabled and others could be ignored, influencing the derivation process and the final result. Therefore we want to extend the language in this sense.

We plan to investigate the complexity of tractable soft constraints classes [7] in order to cast them in a Datalog-based language. Therefore, we want to extend also the RT^C language [13] (based on Datalog enhanced with crisp constraints) in its soft version.

References

1. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *Lecture Notes in Computer Science*. Springer, 2004.
2. S. Bistarelli, F. Martinelli, and F. Santini. Weighted datalog and levels of trust. In *To appear in Advances in Policy Enforcement*. IEEE, 2008.
3. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
4. S. Bistarelli and F. Rossi. Semiring-based constraint logic programming: syntax and semantics. *ACM Trans. Program. Lang. Syst.*, 23(1):1–29, 2001.
5. P. Bonatti, C. Duma, D. Olmedilla, and N. Shahmehri. An integration of reputation-based and policy-based trust management. In *Semantic Web Policy Workshop*, 2005.
6. S. Chakraborty and I. Ray. Trustbac: integrating trust relationships into the rbac model for access control in open systems. In *SACMAT '06: Proc. of Access control models and technologies*, pages 49–58. ACM, 2006.
7. D. A. Cohen, M. C. Cooper, P. G. Jeavons, and A. A. Krokhin. The complexity of soft constraint satisfaction. *Artif. Intell.*, 170(11):983–1016, 2006.
8. M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli. Fine grained access control with trust and reputation management for globus. In *OTM Conferences (2)*, pages 1505–1515, 2007.
9. J. DeTreville. Binder, a logic-based security language. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 105, Washington, DC, USA, 2002. IEEE Computer Society.
10. T. Jim. SD3: A trust management system with certified evaluation. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 106, Washington, DC, USA, 2001. IEEE Computer Society.
11. A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.
12. N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, 2003.
13. N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *PADL '03: Proc. of Practical Aspects of Declarative Languages*, pages 58–73. Springer-Verlag, 2003.
14. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *SP '02: Proc. of Security and Privacy*, page 114. IEEE Computer Society, 2002.
15. F. Martinelli and M. Petrocchi. A uniform approach for the modeling of security and trust on protocols and services. In *ICS '06: International Workshop on Computer Security*, 2006.
16. G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *WiSe '04: Workshop of Wireless security*, pages 1–10. ACM, 2004.