# A Semiring-based Framework for the Deduction/Abduction Reasoning in Access Control with Weighted Credentials

Stefano Bistarelli, Fabio Martinelli, Francesco Santini

# A Semiring-based Framework for the Deduction/Abduction Reasoning in Access Control with Weighted Credentials

Stefano Bistarelli[a,1], Fabio Martinelli[b,2], Francesco Santini[c,1,3,*]

[a] *Dipartimento di Matematica e Informatica, Università di Perugia, Italy*
[b] *Istituto di Informatica e Telematica, CNR, Pisa, Italy*
[c] *Centrum Wiskunde & Informatica, Amsterdam, The Netherlands*

**Abstract**

We present a variant of the *Datalog* language (we call it $Datalog^W$), which is able to deal with weights on ground facts. The weights are chosen from a semiring algebraic structure. Our goal is to use this language as a semantic foundation for trust-management languages, in order to express trust relationships associated with a preference (e.g., a cost, an uncertainty, a trust or a fuzzy value). We apply $Datalog^W$ as the basis to give a uniform semantics to a weighted extension of the *RT* language family, called $RT^W$. Moreover, we show that we can model the deduction and abduction reasoning with semiring-based soft constraints: deduction can validate or not the access request, while abduction can be used to compute the missing credentials if the access is denied and the level of preference that would grant the access.

*Keywords:* Role-based Trust-management Languages, Abduction, Deduction, Soft Constraint.

## 1. Introduction and Motivations

In this work we extend the *Role-based Trust-management* language family (*RT*) [28] with weighted credentials, in order to enrich the access authorization

*Corresponding author
*Email addresses:* bista@dmi.unipg.it (Stefano Bistarelli), fabio.martinelli@iit.cnr.it (Fabio Martinelli), F.Santini@cwi.nl (Francesco Santini)

[1] Stefano Bistarelli is partially supported by Istituto di Informatica e Telematica, CNR, Pisa, Italy. Research partially supported by MIUR PRIN 20089M932N project: "Innovative and multi-disciplinary approaches for constraint and preference reasoning", by CCOS FLOSS project "Software open source per la gestione dell'epigrafia dei corpus di lingue antiche", and by INDAM GNCS project "Fairness, Equità e Linguaggi".

[2] Work partially supported by the FP7 EU projects Aniketos, CONNECT and NESSoS.

[3] This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. This Programme is supported by the Marie Curie Co-funding of Regional, National and International Programmes (COFUND) of the European Commission.

decision with a value that represents an access degree level. The weights set and operations are modeled within a semiring-based framework [2, 8]. These weights can represent a cost, a trust or an uncertainty score, or in general, a preference associated with each credential. As a second step, we show how, by using semiring-based soft constraints [2, 8] and a policy written in the weighted version of $RT$ (i.e., $RT^W$), we can model the *deduction* and *abduction* processes [25, 32], which can be respectively used to deduce if the access can be granted and to compute the missing credentials if it cannot be deduced.

A trust-management language is needed 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 (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 [31, 13, 11]). 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). In literature there are many examples where trust or reputation are computed by aggregating some values together [23], 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 on the authorization decision process.

For this reason, in this paper we describe a weighted version of *Datalog* (we call it $Datalog^W$) where the rules are enhanced with values taken from a proper semiring structure [2, 8], in order to model the preference/cost system.

We use $Datalog^W$ as the basis to give declarative semantics to a weighted version of the $RT$ family of languages [28], used to describe policies and credentials. We called our weighted family $RT^W$ and, according to the guidelines given in [28], it comprehends the languages $RT_0^W$, $RT_1^W$, $RT_2^W$, $RT^{WT}$ and $RT^{WD}$, where the statements are "soft" in the sense that each credential is associated to a semiring value.

Indeed, there are good reasons to prefer a language, as $Datalog^W$, that is declarative and has a formal foundation. In this sense, we are following a similar approach as done in [27] for the $RT$ family, where classical *Datalog* with crisp constraints has been proposed for the $RT$ languages. Since trust is not necessarily crisp, $Datalog^W$ can be used to give a formal "weighted" semantics to the same family of languages instead. Moreover, other languages in literature can benefit from the underlying semantics provided by $Datalog^W$: several trust-management languages are based on *Datalog*, e.g., Delegation Logic [26], the $RT$ [28] (as already stated), KeyNote [10] and RBAC [34].

Our systematic approach to give weights to facts and rules, contributes

also towards bridging the gap between "rule-based" trust management (i.e., crisp security mechanisms) and "reputation based" trust management [23] (i.e., weighted security mechanisms).

Having defined the language, then we need to formally (and then in practice) define the fundamental reasoning processes used to control the access for $RT^W$ policies and credentials. Two basic services can be used inside access control systems: deduction and abduction [25, 32].

In deduction [25, 32], given a policy and a set of additional facts and events, the service finds out all consequences (actions or obligations) of the policy and the facts, i.e., whether granting the access can be deduced from the policy and the current facts. Since we are considering $RT^W$, deduction is improved with a $t$ threshold score, thus allowing the access only if the presented weighted credentials can provide a level better than $t$ (where "better" depends on the chosen semiring).

The second service in access control, requested for more complex reasoning, is abduction [25, 32]. Loosely speaking in reverse, given a policy and a request of access, abduction consists in finding the credentials/events that would grant access, i.e., a (possibly minimal) set of facts that added to the policy would make the request a logical consequence. The intuition behind an interactive (client-server) access control system is the following: *i)* initially a client submits a set of credentials and a service request then, *ii)* the server checks whether the request is granted by the access policy according to the client's set of credentials. If the check fails, *iii)* by using abductive reasoning the server finds a (minimal) solution set of (disclosable) missing credentials that unlocks the desired resource and *iv)* returns them to the client, so that *v)* he can provide them in the second round (however, this kind of negotiation protocol is outside the scope of this paper). Together with the missing credentials, our abduction process returns also the (minimal) missing level of preference that would grant the access, in order to reach the imposed $t$ threshold.

To propose a practical example, access control for autonomic communication needs the abduction reasoning service [35]. The key idea is that in an autonomic network a client may have the right credentials but may not know them and thus an autonomic communication server needs a way to avoid leaving the client stranded.

The deduction and abduction operations on $RT^W$ policies have been implemented by semiring-based soft constraints operators (see Section 2), keeping the underlying machinery of our work within the same framework defined in [2, 8, 6].

In order to show a real implementation of the deduction/abduction procedures over $RT^W$ policies, finally we have used the *Constraint Handling Rules* language (*CHR*) [21]. CHR is a high-level language designed for writing user-defined constraint systems. It is a committed-choice language consisting of guarded rules that rewrite constraints into simpler ones until they are solved.

This paper integrates a polished revision of [5, 6, 7]. In Section 2 we describe the background about semirings and semiring-based soft constraints. In Section 3 we present a weighted version of *Datalog*, i.e., $Datalog^W$, while Section 4 features $RT_0^W$, $RT_1^W$ and $RT_2^W$, based on $Datalog^W$; Section 4 also describes

the semantics inclusions between these three "brick" languages. Then, Section 5 and Section 6 respectively present other two languages of the family, $RT^{WT}$ and $RT^{WD}$, which can be used, together or separately, with each of $RT_0^W$, $RT_1^W$, or $RT_2^W$. Section 7 provides how the deduction reasoning can be modeled with soft constraint operators and presents a CHR implementation; Section 8 shows the same for the abduction reasoning. Section 9 reports the related work in literature and, finally, in Section 10 we draw the final conclusions and ideas about future work.

## 2. Background on Semiring-based Soft Constraints

*Semirings.* An absorptive semiring [4] $S$ can be represented as a $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ tuple such that: *i)* $A$ is a set and $\mathbf{0}, \mathbf{1} \in A$; *ii)* $+$ is commutative, associative and $\mathbf{0}$ is its unit element; *iii)* $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. Moreover, $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leq_S$ over $A$ such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [8]): *i)* $\leq_S$ is a partial order; *ii)* $+$ and $\times$ are monotonic on $\leq_S$; *iii)* $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; *iv)* $\langle A, \leq_S\rangle$ is a complete lattice and, for all $a, b \in A$, $a+b = lub(a, b)$ (where *lub* is the *least upper bound*). Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$ (or simply $a \leq b$ when the semiring will be clear from the context), we will say that *$b$ is better than $a$.*

In [4] the authors extended the semiring structure by adding the notion of *division*, i.e., $\div$, as a weak inverse operation of $\times$. An absorptive semiring $S$ is *invertible* if, for all the elements $a, b \in A$ such that $a \leq b$, there exists an element $c \in A$ such that $b \times c = a$ [4]. If $S$ is absorptive and invertible, then, $S$ is *invertible by residuation* if the set $\{x \in A \mid b \times x = a\}$ admits a maximum for all elements $a, b \in A$ such that $a \leq b$ [4]. Moreover, an absorptive $S$ is *residuated* if the set $\{x \in A \mid b \times x \leq a\}$ admits a maximum for all elements $a, b \in A$: with an abuse of notation, the maximal element among solutions is denoted $a \div b$.

The notion of semiring division can be applied to all the classical semirings instances [8], as the *Boolean* $\langle\{false, true\}, \vee, \wedge, false, true\rangle$, *Fuzzy* $\langle[0, 1], max, min, 0, 1\rangle$, *Probabilistic* $\langle[0, 1], max, \hat{\times}, 0, 1\rangle$ ($\hat{\times}$ is the arithmetic multiplication) and *Weighted* $\langle\mathbb{R}^+ \cup \{+\infty\}, min, \hat{+}, \infty, 0\rangle$ semiring ($\hat{+}$ is the arithmetic plus operation): the reason is that all these semirings are complete and consequently residuated [4]. Therefore, for all these semirings it is possible to use the $\div$ operation as a "particular" inverse of $\times$.

A further (residuated) semiring that will be used in Ex. 3 (see Section 4) is the *path semiring* [36]: $S_{trust} = \langle\langle[0, 1], [0, 1]\rangle, +_p, \times_p, \langle 0, 0\rangle, \langle 1, 1\rangle\rangle$ to compute the trust level, where

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

$$\langle t_i, c_i\rangle \times_p \langle t_j, c_j\rangle = \langle t_i \hat{\times} t_j, c_i \hat{\times} c_j\rangle$$

In this case, trust information is represented by a couple of values $\langle t, c\rangle$: the first component represents a trust value in the range $[0, 1]$, while the second 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. According to the definition of $\times_p$, the trust and confidence values are respectively multiplied together (i.e., $\hat{\times}$ is the arithmetic multiplication), and with $+_p$ we prefer the couple with the higher confidence values; the two confidence values are equal, we choose the couple with the higher trust value.

*Soft Constraints.* A *soft constraint* [8, 2] may be seen as a constraint where each instantiation of its variables has an associated preference. Given $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables $V$ over a (finite) domain $D$, a soft constraint is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring. Using this notation $\mathcal{C} = \eta \to A$ is the set of all possible constraints that can be built starting from $S$, $D$ and $V$.

$\bar{\mathbf{0}}$ and $\bar{\mathbf{1}}$ respectively represent the constraints associating the $\mathbf{0}$ and $\mathbf{1}$ of the semiring to all assignments of domain values; in general, the $\bar{a}$ function returns the semiring value $a$.

Any function in $\mathcal{C}$ involves all the variables in $V$, but we impose that it depends on the assignment of only a finite subset of them. So, for instance, a binary constraint $c_{x,y}$ over variables $x$ and $y$, is a function $c_{x,y} : (V \to D) \to A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$ (the *support*, or *scope*, of the constraint). Note that $c\eta[v := d_1]$ means $c\eta'$ where $\eta'$ is $\eta$ modified with the assignment $v := d_1$. Notice also that, with $c\eta$, the result we obtain is a semiring value, i.e., $c\eta = a$ with $a \in A$.

Given the set $\mathcal{C}$, the combination function of constraints $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times c_2\eta$.

The the constraint division function $\oslash : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ (which subtracts the second constraint from the first one) is instead defined as $(c_1 \oslash c_2)\eta = c_1\eta \div c_2\eta$ [4]: given $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, $a \div b$ is defined as the $m$ maximal element of the set $\{x \in A \mid b \times x \leq a\}$. Therefore, $n \leq_S m$ for every other element $n$ in the set and, extended to constraints, $c_n \sqsubseteq_S c_m \iff c_n(x = n) \leq_S c_m(x = m)$, i.e., the other possible constraints obtained by choosing a different $x$ in $\{x \in A \mid b \times x \leq_S a\}$ imply the one obtained by choosing the $m$ maximal element. For this reason, the $\oslash$ operator is able to find the minimal explanation (see Section 8), since $m$ is the maximum we can remove with $\div$.

$$c_1 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \cup \{+\infty\} \quad \text{s.t. } c_1(x) = x + 3$$

$$c_2 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \cup \{+\infty\} \quad \text{s.t. } c_2(x) = x + 5$$

$$c_3 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \cup \{+\infty\} \quad \text{s.t. } c_3(x) = 2x + 8$$

$$c_4 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \cup \{+\infty\} \quad \text{s.t. } c_4(x) = x + 1$$

$$c_5 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \cup \{+\infty\} \quad \text{s.t. } c_5(x) = x + 7$$

Figure 1: Five weighted soft constraints; notice that $c_3 = c_1 \otimes c_2$ and $c_5 = c_3 \obar c_4$.

It is worth to notice that our $\obar$ operation can be considered as a "relaxation", and not a strict removal of a constraint. For example, considering the five soft constraints reported in Figure 1 and the *Weighted* semiring $S_{Weighted} = \langle \mathbb{R}^+ \cup \{+\infty\}, min, \hat{+}, +\infty, 0 \rangle$, we have that $c_1 \otimes c_2 = c_3$. Since the $\obar$ operator consists in a relaxation, we can also remove $c_4$ from $c_3$ (i.e., not only $c_1$ or $c_2$), that is $c_3 \obar c_4 = c_5$, even if $c_4$ is different from both $c_1$ and $c_2$.

Informally, performing the $\otimes$ or the $\obar$ between two constraints means building a new constraint whose support involves all the variables of the original ones, and which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying or, respectively, dividing the elements associated by the original constraints to the appropriate sub-tuples. The partial order $\leq_S$ over $\mathcal{C}$ can be easily extended among constraints by defining $c_1 \sqsubseteq c_2 \iff c_1\eta \leq c_2\eta$. Moreover, consider the set $\mathcal{C}$ (and its power-set $\wp(\mathcal{C})$) and the partial order $\sqsubseteq$. Then, an entailment relation $\vdash \subseteq \wp(\mathcal{C}) \times \mathcal{C}$ is defined s.t. for each $C \in \wp(\mathcal{C})$ and $c \in \mathcal{C}$, we have $C \vdash c \iff \bigotimes C \sqsubseteq c$.

Given two constraints $c_1$ and $c_2$, if we have $c_1 \vdash c_2$ then we can say that $c_2$ is a *logical consequence* of $c_1$: we will use this operator in order to abduce the missing credentials in Section 8.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of $c$ over $V - \{v\}$, written $c \Downarrow_{(V \setminus \{v\})}$ is the constraint $c'$ such that $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support.

A *Soft Constraint Satisfaction Problem* (*SCSP*) [2] is defined as $P = \langle C, con \rangle$ ($C$ is the set of constraints and $con \subseteq V$, i.e., the subset of interest among the problem variables). The *best level of consistency* notion defined as $blevel(P) = Sol(P) \Downarrow_{\emptyset}$, where $Sol(P) = (\bigotimes C) \Downarrow_{con}$ [2]. A problem $P$ is $\alpha$-consistent if $blevel(P) = \alpha$ [2]; if $P$ is $\alpha_1$-consistent then it is also $\alpha_2$-consistent if $\alpha_1 \leq_S \alpha_2$. $P$ is instead simply "consistent" iff there exists $\alpha >_S \mathbf{0}$ such that $P$ is $\alpha$-consistent [2]. $P$ is inconsistent if it is not consistent.

**Example 1.** *Figure 2 shows a weighted SCSP* $P = \langle C = \{c_1, c_2, c_2\}, con = \{X, Y\} \rangle$ *as a graph: the used semiring is the* Weighted *semiring, i.e.,* $\langle \mathbb{R}^+ \cup \{+\infty\}, \min, \hat{+}, \infty, 0 \rangle$. *Variables and constraints are represented respectively by nodes and by arcs (unary for* $c_1$ *and* $c_3$*, and binary for* $c_2$*), and semiring values are written to the right of each tuple.* $V = \{X, Y\}$ *and* $D = \{a, b\}$. *The solution*

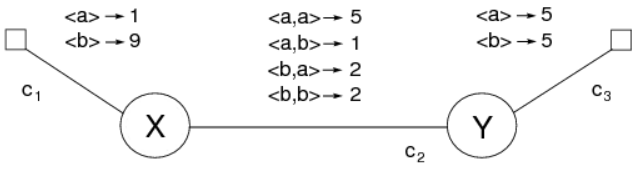


Figure 2: A soft CSP based on a Weighted semiring.

*of the CSP in Figure 2 associates a semiring element to every domain value of variables $X$ and $Y$ by combining all the constraints together, i.e., $Sol(P) = \bigotimes C$. For instance, for the tuple $\langle a, a\rangle$ (that is, $X = Y = a$), we have to compute the sum of* 1 *(which is the value assigned to $X = a$ in constraint $c_1$),* 5 *(which is the value assigned to $\langle X = a, Y = a\rangle$ in $c_2$) and* 5 *(which is the value for $Y = a$ in $c_3$). Hence, the resulting value for this tuple is* 11*. For the other tuples, $\langle a, b\rangle \rightarrow 7$, $\langle b, a\rangle \rightarrow 16$ and $\langle b, b\rangle \rightarrow 16$. The* blevel *for the example in Figure 2 is* 7 *(related to the solution $X = a$, $Y = b$), which represents the solution with the best preference.*

## 3. A Weighted Extension of Datalog

*Datalog* [12] 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:

$$R_0(t_{0,1}, \ldots, t_{0,k_0}) : \text{-} R_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, R_n(t_{n,1}, \ldots, t_{n,k_n})$$

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 the body is empty, the rule is called a fact. A program in *Datalog* is a set of rules; a program $P$ 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:

$$R_i(x_{i,1}, \ldots, x_{i,k_i}) : \text{-} \; s$$

Therefore, the extension is obtained by associating to ground facts a value $s \in A$ taken from the semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$. 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 $\langle \mathbb{R}^+ \cup \{+\infty\}, min, \hat{+}, \infty, 0\rangle$, trying to minimize the overall sum at the same time. Otherwise, we can find the best

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

Figure 3: A simple $Datalog^W$ program.

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 *Probabilistic* semiring $\langle [0,1], \max, \hat{\times}, 0, 1\rangle$.

Figure 3 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 Figure 3) 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.

*Finite Computation Time.* Being the $Datalog^W$ language a subset of the *Soft Constraint Logic Programming* language [9] with no functions, we can can use the results in [9] to prove that, considering a given $Datalog^W$ program, the time for computing the value of any goal for this program is finite and bounded by a constant, as Theorem 1 states:

**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. Moreover, each refutation in $SR(C)$ has length at most $N$, where $N = \sum_{i=1}^{c} b_i$, $c$ is the number of all clauses with head instantiated in all possible ways, and $b$ is the greatest number of atoms in the body of a clause in program $P$.*

PROOF. The proof of the theorem is identical to the one given in [9]. Each simple refutation tree has the property that no two labels in a path from the root to a leaf are the same. Since such labels are clauses plus head instantiations, each path is long at most as the number of all possible clause head instantiations, say $c$. Furthermore, the branching factor of a simple refutation tree depends on the number of atoms in the bodies of the clauses used in the refutation. Thus, the number of nodes of a simple refutation tree, and, consequently, of steps in a simple refutation, is bounded by $\sum_{i=1}^{c} b_i$. Therefore, solving a goal $C$ takes $O(SR(C) \cdot \sum_{i=1}^{c} b_i)$ steps.

## 4. Extending $RT_0$, $RT_1$ and $RT_2$ with Weights

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 Section 3). This family uniformly extends the classical $RT$ family [28] by associating a semiring value to the basic role definition. Therefore, all the following credentials must be parameterized with a chosen $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ 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.

**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., $s \in A$.

**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* [28] 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.

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 chain with more chances to authorize the requester (among all the credentials revealed by her/him).

A credential, in a general definition, is an attestation of qualification, competence, or authority issued to an individual by a third party with the authority or the competence to do so. A policy describes the rules that grant the authorization. As an example, consider the first four rows in Figure 4 as policy rules, and the other rows as (weighted) credential definition.

**Example 2.** *In Figure 4 we show an example of $RT_0^W$, extending the example presented in [28] with weights. To solve the example in Figure 4, we use a* Weighted *semiring $\langle \mathbb{R}^+ \cup \{+\infty\}, \min, \hat{+}, \infty, 0 \rangle$, where the elements in $\mathbb{R}^+ \cup \{+\infty\}$ represents the number of negative feedbacks the credential has received: for example,* `StateU.highMarks` $\longleftarrow$ $\langle$ `Alice, 4` $\rangle$ *in Figure 4 certifies that Alice has obtained high marks for the exams completed at the StateU university (the credential is issued by StateU) and that this credential has received* 4 *negative feedbacks by a system that collects a reputation value for the statements contained in the credentials. Feedbacks can be given by different kinds of entities: e.g., for the considered credential, students, professors, academic tutors. How these values are collected are outside the scope of the presentation.*

This example is focused on bridging the gap between "rule-based" trust management (i.e., crisp security mechanisms) and "reputation based" trust management [23] (i.e., weighted security mechanisms). However, weights can be used in our framework to represent also other kinds of preference, as (money) costs, fuzzy or certainty degrees and all that can be modeled as semiring (see Section 2).

```
EPub.disct ⟵ EPub.preferred ∩ EPub.brightStudent.
EPub.preferred ⟵ EOrg.highBudget ∩ EOrg.oldCustomer.
EPub.brightStudent ⟵ EPub.goodUniversity.highMarks.
EPub.goodUniversity ⟵ ABU.accredited.
ABU.accredited ⟵ ⟨ StateU, 2 ⟩.
StateU.highMarks ⟵ ⟨ Alice, 4 ⟩.
EOrg.highBudget ⟵ ⟨ Alice, 3 ⟩.
EOrg.oldCustomer ⟵ ⟨ Alice, 2 ⟩.
```

Figure 4: An example in $RT_0^W$, with fuzzy values associated to the credentials.

The example in Figure 4 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 score to both these two credentials to quantify the money cost of these certificates. 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 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. By composing the reputation values of all the credentials in order to be authorized for the discount, we know that the credentials have received a total of $2\hat{+}4\hat{+}3\hat{+}2 = 11$ negative feedbacks. If the access system imposes a threshold of, e.g., 12 negative feedbacks, we can access to the discount since $12 \leq_S 11$, where $S = \langle \mathbb{R}^+ \cup \{+\infty\}, \min, \hat{+}, \infty, 0 \rangle$ in this case.

**Example 3.** *In Figure 5 we extend the example of Figure 4 in order to represent also a case where the access to the system is granted by following different subsets of credentials (i.e., following several refutation paths). With respect to Figure 4, in Figure 5 a customer receives the discount even if she/he presents a good recommendation letter (i.e.,* EPub.famousProf.goodRecLetter*).*

*In this example we use the* path semiring *presented in Section 2, thus a semiring value consists in a couple of* $\langle trust, confidence \rangle$ *feedbacks. The best derivation chain corresponds to the criteria defined by the* $+_p$*: between two couples, we prefer the one with the higher confidence value or, if the confidence values are equal, the couple with the higher trust value. Confidence and trust values are both in the* $[0..1]$ *interval and are composed by using the arithmetic multiplication. Therefore, with the program in Figure 5 we obtain two solution with costs* $\langle 0.3, 0.25 \rangle$ *and* $\langle 0.81, 0.72 \rangle$ *respectively. By using* $+_p$ *we prefer the solution with cost* $\langle 0.81, 0.72 \rangle$.

```
EPub.disct ⟵ EPub.preferred ∩ EPub.brightStudent.
EPub.disct ⟵ EOrg.famousProf.goodRecLetter.
EPub.preferred ⟵ EOrg.highBudget ∩ EOrg.oldCustomer.
EPub.brightStudent ⟵ EPub.goodUniversity.highMarks.
EPub.goodUniversity ⟵ ABU.accredited.
EOrg.famousProf ⟵ ⟨ ProfX, ⟨ 0.9, 0.9 ⟩⟩.
ProfX.goodRecLetter ⟵ ⟨ Alice, ⟨ 0.9, 0.8 ⟩⟩.
ABU.accredited ⟵ ⟨ StateU, ⟨ 0.9, 0.8 ⟩⟩.
StateU.highMarks ⟵ ⟨ Alice, ⟨ 0.8, 0.9 ⟩⟩.
EOrg.highBudget ⟵ ⟨ Alice, ⟨ 0.6, 0.5 ⟩⟩.
EOrg.oldCustomer ⟵ ⟨ Alice, ⟨ 0.7, 0.7⟩⟩.
```

Figure 5: An extension of the example in Figure 4, by using the *path semiring*.

*Figures 6 and Figure 7 show the two Datalog refutation paths for the* $RT_0^W$ *program in Figure 5.*

In the next subsections we suggest how to extend $RT_0^W$ with parameterized roles (thus obtaining $RT_1^W$), and how to further extend the language with logical

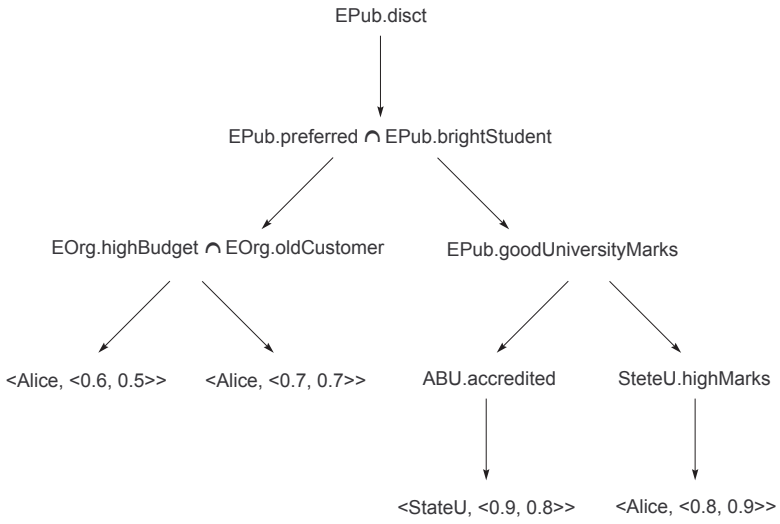


Figure 6: The trust/confidence value for this set of credentials is $\langle 0.3, 0.25 \rangle$ by using the path semiring in Section 3.

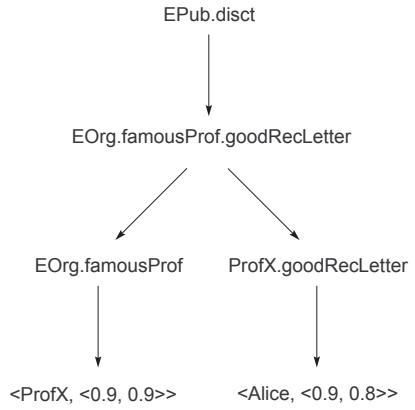


Figure 7: The trust/confidence value for this set of credentials is $\langle 0.81, 0.72 \rangle$ by using the path semiring in Section 3.

rights (obtaining $RT_2^W$). In Section 4.3 we finally show a theorem explaining the inclusion relationships among $RT_0^W$, $RT_1^W$, $RT_2^W$ and their original (i.e., not weighted) versions.

### *4.1. $RT_1^W$: Parametrized Roles*

It is easy to extend $RT_0^W$ in order to enhance it with parameterized roles, thus obtaining a $RT_1^W$ language following the hierarchy presented in [28]. This parametrization can be used to represent the relationships among entities, e.g., *University.professorOf(student)* to name the professor of a student. Parametrization can also be used 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 $\langle A.R(h_1, \ldots, h_n), s\rangle \longleftarrow B$, 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 [28] (see Section 10).

A variable that appears in an $RT_1$ [28], and consequently $RT_1^W$, credential can be either named or anonymous. A named variable takes the form of a question mark "?" followed by an alpha-numeric string. If a variable appears only once in a credential, it does not need to have a name and can be anonymous. An anonymous variable is represented by the question mark alone. Note that two different appearances of "?" in a credential represent two distinct variables [28].

**Example 4.** *In the example in Figure 8 we show an example where* Alice *may receive the discount because StateU released a diploma with her name in* 1962 *and the discount is offered to all the alumni enrolled in the years 1960-1965. Moreover, a proof from a professor of* Alice *must be collected as a credential. Credentials are weighted with negative feedbacks as in Ex. 4. Notice that we can express some constraints on the values of the attributes, that is "*: [1960..1965]*" [28].*

```
                EPub.disct ⟵ EOrg.specYear.
  EOrg.specYear ⟵ StateU.oldAlumni(?X) ∩ StateU.profOf(?X).
StateU.oldAlumni(?X) ⟵ StateU.diploma(?X, ?Year:[1960..1965].
            StateU.profOf(Alice) ⟵ ⟨ ProfX, 3 ⟩.
         StateU.diploma(Alice, 1962) ⟵ ⟨ Alice, 2 ⟩.
```

Figure 8: An example in $RT_1^W$.

### *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. $RT_2^W$ extends $RT_1^W$ with logical rights.

As proposed in [28], 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 $\tau$, 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 $\tau$ [28]. An o-set-definition credential is similar to the role definition credential that we have defined in Section 4.1 for $RT_1^W$: the difference is that the members of o-sets are objects that are not entities. For example, $\texttt{StateURegistrar.cv(modify)} \longleftarrow \langle \texttt{studyPlan}, 0.3 \rangle$ states that the registrar's office of *StateU* university grants to the *studyPlan* of a student the permission to be *modified* only for the 30% of it.

O-set-definition credentials can be translated in *Datalog* exactly as proposed for $RT_0^W$ and $RT_1^W$ in Section 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 $\tau$, while the body can be a value of base type $\tau$, another o-set $B.O(s_1, \ldots, s_m)$ of base type $\tau$, a linked o-set $A.R(t_1, \ldots, t_l).O(s_1, \ldots, s_m)$, in which $R(t_1, ..., t_l)$ is a role name and $O(s_1, \ldots, s_m)$ is an o-set name of base type $\tau$, or an intersection of $k$ o-sets of the base type $\tau$, i.e., $O_1(\ldots) \cap O_2(\ldots) \cap \cdots \cap O_k(\ldots)$ (see Section 5 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. $RT_1^W$ ones), please refer to [28].

**Example 5.** *In the example in Figure 9, we suppose that the plan of study (i.e.,* StudyPlan*) in Computer Science (*cs*) of a given student, part of her/his cv, can be modified for its* 30%*. Moreover, the registrar's office of the StateU university (i.e.,* StudentRegistrar*) states that* Alice *is at her first year at StateU and, therefore, she can modify her plan of study in Computer Science (*cs*) for its* 10% *only, since she is a freshwoman. In this way it is possible to combine the general and more specific rights together; we suppose to use a fuzzy composition of rights by using the* Fuzzy *semiring* $\langle [0,1], max, min, 0, 1 \rangle$*. From the* $RT_2^W$ *in Figure 9 we derive* $\texttt{StudentRegistrar.studyP(modify, studyPlan)} \longleftarrow \langle \texttt{Alice}, 0.1 \rangle$*:* Alice *can modify her* StudyPlan *for a a percentage of* $\min(0.3, 0.1) = 0.1$.

$$\texttt{StudentRegistrar.studyP(modify, ?F : StudentRegistrar.cv(?X))} \longleftarrow \texttt{StudentRegistrar.firstYear(?X)}.$$
$$\texttt{StudentRegistrar.cv(cs)} \longleftarrow \langle \texttt{StudyPlan}, 0.3 \rangle.$$
$$\texttt{StudentRegistrar.firstYear(cs)} \longleftarrow \langle \texttt{Alice}, 0.1 \rangle.$$

Figure 9: An example in $RT_2^W$.

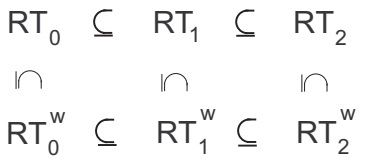


Figure 10: A hierarchy of $RT^W$ languages, compared with the classical $RT$ one.

*4.3. Language Family Inclusion*

In the next theorem we claim that our weighted language family can be used to represent also classical $RT$ credentials [28]. In this sense, the $RT^W$ languages can be considered as a foundation layer for all the classical $RT$ languages.

**Theorem 2 (Language Family Inclusion).** *For each set of statements in the $RT_0$, $RT_1$ or $RT_2$ language, we can find an equivalent set of statements respectively represented in $RT_0^W$, $RT_1^W$ or $RT_2^W$, whose semantics is the same.*

Proof. In Figure 10 we show the theorem result, i.e., the vertical inclusions; the horizontal ones are explained in [28]. The proof of the theorem comes from using the *Boolean* semiring $\langle \{false, true\}, \vee, \wedge, false, true \rangle$ 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 derivation paths, as the corresponding $RT$ set of statements would do. Therefore, the original "crisp" version of the $RT$ languages can be modeled with $RT^W$ languages based on the semiring structure. By using other types of semiring (e.g., the *Weighted* semiring) we clearly become more expressive, since we can take decisions based on different preferences.

In Section 5 and Section 6 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 $i = 0, 1, 2$.

## 5. $RT^{WT}$: Threshold and Separation-of-Duty Policies

$RT^{WT}$ can be used, together or separately, with each of $RT_0^W$, $RT_1^W$, or $RT_2^W$. The resulting combinations are written $RT_0^{WT}$, $RT_1^{WT}$ and $RT_2^{WT}$.

*Threshold* structures are satisfied by the agreement of $k$ out of a set of entities that satisfy a specified condition, while *separation of duty* 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 Section 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 only if two out of three professors confirm this, can be represented by the three rules $Uni.bS \longleftarrow$

$P_1.bS \cap P_2.bS$, $Uni.bS \longleftarrow P_1.bS \cap P_3.bS$ and $Uni.bS \longleftarrow P_2.bS \cap P3.bS$ (where $P_i$ is the $i$th professor and $bS$ means "bright student").

However, with the intersection operator we are not able to express complex policies, for example we can represent the fact that $A$ states that an entity $b$ has attribute $R$ if two different entities having attribute $R_1$ state that $b$ has attribute $R$; for this reason we introduce the new $RT^{WT}$ language. In order to attain this, we need to work with sets of entities. More specifically, $RT^{WT}$ adds to the "unweighted" $RT^{W}$ language the notion of *manifold roles*, which generalizes the notion of roles [28]. 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.

In $RT^{WT}$ we introduce two more types of credentials w.r.t. Section 4:

**Rule 5** $A.R \longleftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$. As we have introduced before with words, the meaning of this credential is $members(A.R) \supseteq members(B_1.R_1 \odot \cdots \odot B_k.R_k) = \{s_1 \cup \cdots \cup s_k | s_i \in members(B_i.R_i)$ 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 the clause is then composed as $w_1 \times w_2 \times \cdots \times w_k$, where $\times$ depends from the chosen $S$ 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 $members(A.R) \supseteq members(B_1.R_1 \otimes \cdots \otimes B_k.R_k) = \{s_1 \cup \cdots \cup s_k | (s_i \in members(B_i.R_i) \wedge s_i \cap s_j = \emptyset)$ 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 the clause is then composed as $w_1 \times w_2 \times \cdots \times w_k$.

As usual, the $Datalog^W$ engine selects the best derivation chain according to the + operator of the semiring. Considering $RT^{WT}$, the translation to $Datalog^W$ rules for **Rule 1**, **Rule 2** and **Rule 4** credentials is the same as the one presented in Section 4. Considering **Rule 3**, **Rule 5** and **Rule 6**, 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 [28] 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 a 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 $1 \leq i \neq j \leq k, s_i \cap s_j = \emptyset$

$$\texttt{StateU.bS} \longleftarrow \texttt{StateU.evaluators.bS}.$$

$$\texttt{StateU.evaluators} \longleftarrow \texttt{StateU.evalProfs} \odot \texttt{StateU.evalExtAdvisor}.$$

$$\texttt{StateU.evalProfs} \longleftarrow \texttt{StateU.evalProf} \otimes \texttt{StateU.evalProf}.$$

$$\texttt{StateU.evalExtAdvisor} \longleftarrow \langle \texttt{A}, 0.9 \rangle. \quad \texttt{StateU.evalExtAdvisor} \longleftarrow \langle \texttt{B}, 0.7 \rangle.$$

$$\texttt{StateU.evalProf} \longleftarrow \langle \texttt{A}, 0.8 \rangle. \quad \texttt{StateU.evalProf} \longleftarrow \langle \texttt{C}, 0.8 \rangle.$$

$$\texttt{StateU.evalProf} \longleftarrow \langle \texttt{D}, 0.6 \rangle.$$

Figure 11: An example in $RT^{WT}$.

**Example 6.** *Suppose that for a university office a student is "bright" (i.e., StateU.bS) if one member of StateU.evalExtAdvisor (i.e., an external advisor, as A and B) and two different members of StateU.EvalProf (i.e., a professor who teaches in that university, as A (again), C and D), confirm this. This can be represented using the $RT^{WT}$ program in Figure 11.*

*If we adopt the* Fuzzy *semiring $\langle [0,1], max, min, 0, 1 \rangle$ representing a fuzzy truth score of the credentials, 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 chain is selected (by using the* max *operator).*

## 6. $RT^{WD}$: Delegation of Role Activations

As $RT^{WT}$ (see Section 5), also $RT^{WD}$ can be used, together or separately, with each of $RT_0^W$, $RT_1^W$, or $RT_2^W$. The resulting combinations are written $RT_0^{WD}$, $RT_1^{WD}$ and $RT_2^{WD}$.

$RT^{WD}$ is finally added to our weighted language 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 new exam results of students, but only having the rights to see her/his number of tickets for. With a weighted extension we are now able to state "how much" can be delegated to another entity, for example a session or a process. This quantitative delegation can be used, for example, to add a numerical information on TM systems as KeyNote [20].

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

$$\texttt{RGroup.projectAccepted} \longleftarrow \texttt{RGroup.submit} \otimes \texttt{RGroup.approve}.$$
$$\texttt{RGroup.submit} \longleftarrow \texttt{RGroup.member}.$$
$$\texttt{RGroup.approve} \longleftarrow \texttt{RGroup.senior}.$$
$$\texttt{RGroup.member} \longleftarrow \texttt{RGroup.senior}.$$

Figure 12: An example in $RT^{WD}$.

than the rights held by the delegating entity (e.g., 100%); this is the considerable improvement obtained by adding a semantic sublayer of weights. 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 [28]: $forRole(B, D, A.R)$ 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: $\texttt{B}_1 \xrightarrow{\texttt{D as A.R}} \texttt{B}_2$ $forRole(B_2, D, A.R) \longleftarrow forRole(B_1, D, A.R)$. This rule means that $B_2$ is acting for "$D$ as $A.R$" if $B_1$ is doing so. Other kinds of delegation rules that can be formulated are presented in [28].

Clearly, even the other rules presented in Section 4 and Section 5 (since $RT^{WD}$ could be see as an extension of $RT^{WT}$) must be modified according to the introduction of the *forRole* predicate. For example, $\texttt{A.R} \longleftarrow \langle \texttt{D}, \texttt{s} \rangle$ becomes $forRole(D, D, A.R)$, *rule_weight*, where *rule_weight* :- $s$ and $s$ is the associated semiring value. 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 [28], and **Rule 1** is the most important rule since it is the "directly weighted" one. 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 $\texttt{B}_1 \xrightarrow{\texttt{D as A.R}} req$ 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 7.** *In this example we show an example of delegation. We use the* Weighted *semiring* $\langle \mathbb{R}^+ \cup \{+\infty\}, min, +, \infty, 0 \rangle$ *with the same meaning used in Ex. 2: weights represent negative feedbacks collected for a credential.*

*In Figure 12 we have a research group (*RGroup*) in which we can find members (*RGroup.member*) and senior members (*RGroup.senior*). Each member can submit a research proposal for the group (*RGroup.submit*), but the request can be accepted (*RGroup.approve*) only by a senior member. Clearly, a senior member is also a member of the group. The $RT^{WD}$ program avoid the acceptance of a project approved by the same senior member who submitted it.*

*Suppose now that A and B are both senior members of the group and have the following credentials:* $\texttt{RGroup.senior} \longleftarrow \langle \texttt{A}, 0 \rangle$ *and* $\texttt{RGroup.senior} \longleftarrow \langle \texttt{B}, 0 \rangle$

*No negative feedbacks have been received for these credentials.*

*Therefore, A can submit a project of the group (variable projectID) by issuing* $\mathtt{A} \xrightarrow{\mathtt{A\ as\ RGroup.member}} \langle \mathtt{proj}(\mathtt{projID}), 2\rangle$*, and B can accept it by issuing* $\mathtt{B} \xrightarrow{\mathtt{B\ as\ RGroup.approve}} \langle \mathtt{proj}(\mathtt{projID}), 1\rangle$*. We can prove that forRole(ReqID,* $\{A, B\}$*, RGroup.projectAccepted), where ReqID is the dummy principal representing proj(projID). If B does not issue the above approval and A proves the order by also issuing* $\mathtt{A} \xrightarrow{\mathtt{A\ as\ RGroup.approve}} \langle \mathtt{proj}(\mathtt{projID}), 2\rangle$*, we cannot prove forRole(ReqID,* $\{A, B\}$*, RGroup.projectAccepted).*

## 7. Deduction with Soft Constraints in Access Control Systems

In this section we show how the $RT^W$ languages, defined with the help of the semiring-based framework presented in Section 2, can be used to add quantitative information to the access control decision.

Deductive reasoning constructs or evaluates deductive arguments. Deductive arguments are attempts to show that a conclusion necessarily follows from a set of premises or hypotheses. Deduction is the basic reasoning in access control system [25], since it checks, from the credentials and the access policy (the premises) if the access authorization can be granted or not (the conclusion). The deduction process [25, 32] can be defined by using the constraint operators shown in Section 2.

**Definition 1.** [Deduction with soft constraints] *Given a soft constraint* $\sigma$ *which represents the current knowledge and a soft constraint c, if* $\sigma \vdash r$ *then* $\sigma$ *entails (i.e., deduces)* $r$*.*

With respect to Definition 1, $\sigma$ represents the policy $p$ and the collected credentials $c$, i.e., $\sigma = p \otimes c$, and $\sigma$ entails (i.e., deduces) the access request $r$ if $\sigma \vdash r$.

*Deduction in CHR.* CHR [21] is a high-level language designed for writing user-defined constraint systems. It is essentially a committed-choice language consisting of guarded rules that rewrite constraints into simpler ones until they are solved. There are three kinds of CHR rules: *simplification*, *propagation* and *simpagation*. Simplification rules replace user-defined constraints by simpler ones. Propagation rules add new redundant constraints that may be necessary to do further simplifications. A simpagation rule is equivalent to a simplification rule with some of the heads repeated in the body. On a simpagation rule only the heads after the \ sign are removed. CHR libraries are available in most Prolog implementations: in particular, for the following examples we have used *SWI-Prolog* [38].

In Figure 13 we implement the deduction process by translating the $RT_0^W$ program in Figure 4. Given the five simplification rules (i.e., *policy*1, ..., *policy*4), which represent the policy, and the first query, (i.e., 1 ?- in Figure 13), that is *accredited(aBU, stateU, 2), highMarks(stateU, alice, 4), highBudget(eOrg, alice,*

```
:- use_module(library(chr)).

:- chr_constraint preferred/2, brightStudent/2, highBudget/3,
                  oldCustomer/3, highMarks/3, goodUniversity/2,
                  accredited/3, discount/2, access/1.

policyrule1   @ preferred(X, A), brightStudent(X, B), discount(X, T) <=>
                (A+B)<= T | access(X).
policyrule2   @ highBudget(Z, X, A), oldCustomer(Z, X, B) <=>
                preferred(X, C), C is (A+B).
policyrule3   @ goodUniversity(X, A), highMarks(X, Y, B) <=>
                brightStudent(Y, C), C is (A+B).
policyrule4   @ accredited(X, Y, A) <=> goodUniversity(Y, A).

1 ?- accredited(aBU, stateU, 2), highMarks(stateU, alice,4 ),
     highBudget(eOrg, alice, 3), oldCustomer(eOrg, alice,2 ),
     discount(alice, 12).

access(alice)

2 ?- accredited(aBU, stateU, 2), highMarks(stateU, alice, 4),
     highBudget(eOrg, alice, 3), discount(alice, 12).

brightStudent(alice, 6 )
highBudget(eOrg, alice, 3)
discount(alice, 12)
```

Figure 13: Deduction with CHR rules: solving the example in Figure 4.

*3), oldCustomer(eOrg, alice, 2), discount(alice, 12)*) which represents the set of presented weighted credentials and the access request (i.e., *discount(alice, 12)*), we are able to deduce that Alice can access to the discount. In fact the system answers with *access(alice)*. The solver can fire the first rule (i.e., *policyrule1*) because the sum of the negative feedbacks of the credentials, which is 11, is lower than the requested threshold in the discount request, which is 12. Notice that lower means better in this case, since in Ex. 2 we consider the *Weighted* semiring $S = \langle \mathbb{R}^+ \cup \{+\infty\}, \min, \hat{+}, \infty, 0 \rangle$.

The implementation in Figure 13 finds if the corresponding SCSP problem is $T$-consistent ($T$ is in *policyrule1*, see Section 2 for $\alpha$-consistency), where $T$ is represented by the access threshold, determined by *discount*. For query 1 ?- in Figure 13 we find out that the problem is 12-consistent, since the sum of the credentials is 11 and $12 \leq_S 11$.

The second query, that is 2 ?- in Figure 13, is presented to show what happens when some necessary credentials are not presented to the authorization system: without the *oldCustomer(eOrg, alice, 2)* credential missing in the query, the CHR rules in Figure 13 are not able to fire *policyrule2* rule. The query returns what can be computed with *policyrule3* and *policyrule4* rule.

## 8. Abduction of Weighted Credentials with Soft Constraints

In this Section we define the abduction process [32] by using the semiring-based framework presented in Section 2. In particular, we adopt the $\oplus$ and $\vdash$

operators (see Section 2 for their formal definitions).

Abduction allows inferring $a$ as an explanation of $b$. Because of this, abduction allows the precondition $a$ to be abduced from the consequence $b$ [37, 32]. For example suppose that, "I received a bad review for my paper on a Chemistry-related topic"; but if "I have no background on Chemistry" then the first statement would not be so surprising. Therefore, by abductive reasoning, the possibility that "I have no background on Chemistry" is reasonable and a good explanation for "I received a bad review for my paper on a Chemistry-related topic". Formally, in logic explanation is done from a logical theory $T$ representing a domain and a set of observations $O$ [37]. Abduction is the process of deriving a set of explanations of $O$ according to $T$ and choosing one of those explanations. For $E$ to be an explanation of $O$ according to $T$, it should satisfy two conditions: *i)* $O$ follows from $E$ and $T$, and *ii)* $E$ is consistent with $T$ [37].

Therefore, if $\sigma$ does not store enough information to satisfy $c$ (i.e., the $c$ conclusion cannot be obtained with deduction), it is interesting to automatically obtain the missing information that would allow to satisfy $c$. In Definition 2 we define abduction by using the $\oslash$ operation presented in Section 2:

**Definition 2.** [Abduction with soft constraints] *Given two soft constraints $\sigma$ and $r$, the abduction process is aimed at finding a constraint $d$, such that $(\sigma \otimes d) \vdash r$, i.e., $d = r \oslash \sigma$.*

Notice that the division operator we are adopting is a deterministic operator (see Section 2) and provides a single solution $d$ (which is also the minimal solution, as explained in the following of this section). Considering Definition 2, the trivial case is when $\sigma \vdash r$ since $r \oslash \sigma = \bar{\mathbf{1}}$, that is nothing must be added to $\sigma$ in order to satisfy $r$, because it is already implied. If we consider our access control scenario with policies and credentials written in $RT^W$, $\sigma = p \otimes c$ represents the policy $p$ and the $c$ set of credentials, and the abduced $d$ constraint is such that $(\sigma \otimes d) \vdash r$, i.e., $d = r \oslash \sigma$, where $d$ is the abduced credential for the access request $r$.

An explanation is an input which can explain the result (via the given the inference rules). A minimal explanation, is one which makes just as many assumptions as needed to obtain an explanation [16]. The concept of minimal explanation is important because it consists in the minimal amount of information needed to obtain the desired consequence: disclosing more credentials could lead to privacy problems since more than enough information is revealed. The abduction operation in Definition 2 finds the minimal explanation as proposed in [16] for the crisp version, according to the definition of $\div$ and $\oslash$ given in Section 2. Therefore, the $d$ constraint in Definition 2 is the minimal explanation of $r$.

Suppose to have three constraints *Student(x)*, *HighMarks(x)* and *Access(x)* which define a preference for a given $x$ student: the support of these three constraints is $\{x\}$. If we have that

$$\sigma = Student(Alice)$$

$$r = \mathit{Access}(\mathit{Alice})$$

and

$$\mathit{Student}(\mathit{Alice}) \otimes \mathit{HighMarks}(\mathit{Alice}) \vdash \mathit{Access}(\mathit{Alice})$$

then $d = r \oslash \sigma = \mathit{HighMarks}(\mathit{Alice})$.

If we use the $S_{Fuzzy} = \langle [0,1], max, min, 0, 1\rangle$ semiring, if $\mathit{Student}(\mathit{Alice}) = 0.9$ and $\mathit{Access}(\mathit{Alice}) = 0.7$, then $\mathit{HighMarks}(\mathit{Alice}) = 0.7$. Thus Alice needs to attest the fact that she has high marks with a trust score 0.7 or greater. In the *Fuzzy* semiring, when $\sigma(x) \leq r(x)$, then $d(x) = 1$, otherwise $d(x) = r(x)$, according to the definition of $\div$ given in [4]:

$$a \div b = max\{x \mid min\{b, x\} \leq a\} = \begin{cases} 1 & \text{if } b \leq a \\ a & \text{if } a < b \end{cases}$$

Now suppose to consider the $S_{Weighted} = \langle \mathbb{R}^+ \cup \{+\infty\}, min, \hat{+}, +\infty, 0\rangle$ semiring. If $\sigma = \mathit{Student}(\mathit{Alice}) = 7$ and $r = \mathit{Access}(\mathit{Alice}) = 10$, then $r \oslash \sigma = d = \mathit{HighMarks}(\mathit{Alice}) = 3$. Therefore, this abduction operation suggests Alice to provide a credential attesting that she has *HighMarks* with a preference of 3 (e.g. votes) at least. The definition of $\div$ (i.e., the arithmetic subtraction $\hat{-}$) is:

$$a \div b = min\{x \mid b \hat{+} x \geq a\} = \begin{cases} 0 & \text{if } b \geq a \\ a \hat{-} b & \text{if } a > b \end{cases}$$

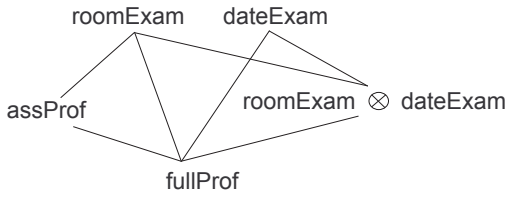


Figure 14: A lattice representing the strength of roles: *fullProf* entails all the other roles.

Notice that the credentials can be seen as partially ordered: *assistant professor* < *associate professor* < *full professor*, that is the *full professor* credential provides higher access capabilities w.r.t. *assistant professor*. This partial order can be easily represented with semiring (see Sec 2). This role hierarchy, where higher the role in the hierarchy, more powerful it is, is adopted also in [25]. A role dominates another role if it is higher in the hierarchy and there is a direct path between them in the lattice representing the hierarchy of roles. For example, $\mathit{fullProf}(x) \vdash \mathit{dateExam}(x) \otimes \mathit{roomExam}(x)$, but $\mathit{assProf}(x) \vdash \mathit{roomExam}(x)$ only, i.e., the assistant professor has less rights. The complete lattice of roles is shown in Figure 14.

```
:- use_module(library(chr)).

:- chr_constraint preferred/2, brightStudent/2, highBudget/3, oldCustomer/3,
                  discount/2, sum/1.


abdrule1   @   discount(X, A), brightStudent(X, B)  <=> C is (A-B),
               preferred(X, C).
abdrule2   @   discount(X, A) <=> preferred(X, A).
abdrule3   @   preferred(X, A), highBudget(Z, X, B) <=> C is (A-B),
               oldCustomer(Z, X, C).
abdrule4   @   preferred(X, A), oldCustomer(Z, X, B) <=> C is (A-B),
               highBudget(Z, X, C).
abdrule5   @   preferred(X, A) <=> oldCustomer(Z, X, B),
               highBudget(Z, X, C) , sum(A).

1 ?- brightStudent(alice, 6), highBudget(eOrg, alice, 3), discount(alice, 12).

oldCustomer(eOrg, alice, 3)

2 ?- discount(alice, 12).

highBudget(_G88354, alice, _G88403)
oldCustomer(_G88354, alice, _G88356)
comb(12)
```

Figure 15: Abduction with CHR rules.

*Abduction in CHR.* In Figure 15 we implement in CHR the abduction procedure for the $RT_0^W$ example presented in Figure 4.

When the *access* constraint is produced by the program in Figure 13, in that case we deduce that the access is not granted. At this point, the rules in Figure 15 can be used to abduce the missing credentials with the maximum level possible of negative feedbacks. The query is now represented by the output obtained at the end of the failed deduction procedure in Figure 13, i.e., $brightStudent(alice, 6), highBudget(eOrg, alice, 3), discount(alice, 12)$. The answer to this query (i.e., 1 ?- in Figure 15) is that we need the *oldCustomer* credential with a number of negative feedbacks less or equal than 2 in order to receive the access permission: the sum of the weights of the two statements (i.e., *brightStudent(alice, 6), highBudget(eOrg, alice, 3)*) is 9 and the discount is granted below a threshold of 12.

Notice that, w.r.t. the policy in Figure 4, in Figure 15 we show only the subset of the abduction rules necessary to compute an answer to query 1 ?- in Figure 15: this subset is able to deal with the branch under *EPub.preferred*, thus the *EPub.brightStudent* branch misses from Figure 15. These rules are not necessary four our sake, because the *brightStudent* has been already deduced by the program in Figure 13. This can be practically seen from the the second query in Figure 15 (i.e., 2 ?-), where, providing only *discount(alice, 12)*, the program abduces only *oldCustomer* and *highBudget*. Moreover, it replays with a further constraint stating that the sum of negative feedbacks for these credentials must be 12 at most (i.e., *comb(12)*).

## 9. Related Work

Similar variations for *RT* family languages are defined and implemented in [31] by using different formal tools. There, an initial comparison (and integration) between role-based trust-management (e.g., *RT*) and reputation-based trust systems has been performed and a preliminary (ad-hoc) implementation of the weighted *RT* language is presented in [18] for GRID systems. However, having a uniform semantics approach, as the weighted datalog we are advocating here, to model these languages could be very useful to provide a common understanding as well as a basis for systematic comparison and uniform implementation.

An important extension that significantly enhances the expressivity of *RT* is represented by $RT^C$ [27]. 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. Our aim is instead the representation of trust levels modeling preferences for credentials. Therefore, our extension is completely orthogonal w.r.t. $RT^C$.

Some TM systems, such as KeyNote [10] and SPKI/SDI [15], use credentials to delegate permissions, but their delegation structure is too limited. Each credential delegates certain permissions from its issuer to its subject, acting like a capability, and therefore they do not address the distributed nature of authority in distributed systems. For instance, it is not possible to express the statement that anyone who is a student is entitled to a discount.

*Datalog* was originally developed as a query/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 [26], the *RT* framework [28] (as discussed in this paper), SD3 (Secure Dynamically Distributed Datalog) [22] and Binder [19]. 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.

Several approaches advocated the usage of trust levels w.r.t. attributes, also stated directly in digital credentials. In [11], 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 [13]. All these works use specific logics and approaches.

In [1] the authors represent access policies as (crisp) constraint logic programs and apply this model to the *Role-Based Access Control* language [34] (RBAC), which is clearly related to *RT*. In that work, temporal forms of authorization are used to specify with constraints an interval of time for which an authorization is to hold. Our approach considers instead "weighted" credentials to represent uncertainty or fuzziness (for instance), instead of crisp intervals of values.

As far as we know, deduction and abduction operations has never been tied to the use of weighted credentials. The importance of crisp deduction/abduction reasoning for policy-based management of autonomic networks is highlighted in [25]. An Autonomic Network [25] crosses organizational boundaries and is

provided by entities that see each other just as business partners. Policy-based network management already requires a paradigm shift in the access control mechanism, from identity-based access control to trust management and negotiation. In [25] the authors present an algorithm whose operations are expressed with logic, and so the access authorization procedure is not directly implemented with a tool in the paper.

Among the most noticeable works concerning logical reasoning and (crisp) constraints, we need to cite Maher, e.g., [29] where he investigates abduction applied to fully-defined predicates, specifically linear arithmetic constraints over the real numbers. In [30], Maher and Huang address the problem of computing and representing answers of constraint abduction problems over the Herbrand domain. This problem is of interest when performing type inference involving generalized algebraic data types. Notice that in [33] the authors present a CHR-based tool for detecting security policy inconsistencies, where (not-weighted) policies are represented by CHR rules.

Concerning instead systems to implement abduction and deduction processes, abduction reasoning has been already realized in *HYPROLOG* [14] (available also in SWI-Prolog), which is an extension of Prolog and CHR with abduction and assumptions. The system is basically implemented by a compiler that translates the HYPROLOG syntax in a rather direct way into Prolog and CHR.

A different implementation is represented by ACLP [24]; ACLP is a system which combines abductive reasoning and constraint solving by integrating the frameworks of *Abductive Logic Programming* and *Constraint Logic Programming* (CLP). The ACLP system is currently implemented on top of the CLP language of ECLiPSe as a meta-interpreter exploiting its underlying constraint solver for finite domains.

## 10. Conclusion and Future Work

We have proposed an extension of the $RT$ family languages called $RT^W$, whose semantics is based on a weighted extension of *Datalog* (i.e., $Datalog^W$), obtained by representing weights with semiring algebraic structures (see Section 2). These languages can be used to deal also with vague and imprecise trust level of credentials during the access control decision: in general, a preference or a cost can be associated to each credential. In practice, we can manage and combine together different levels of truth, costs or (in general) preferences and finally have a more informative feedback value on to authorize the access request or not.

We have extended the $RT$ family of languages [28] and we have shown that classical $RT_0$ and $RT_1$, $RT_2$ are respectively included in our $RT_0^W$, $RT_1^W$ and $RT_2^W$ extensions. Moreover, we have extended other two languages of the RT family, consequently obtaining $RT^{WT}$ and $RT^{WD}$. After that, we have shown how both deduction and abduction reasoning over $RT^W$ policies can be modeled with semiring-based soft constraint operators (see Sec 2); in this way, within the same semiring-based framework, it is possible to deduce if a given set of

weighted credentials has enough cost/trust/preference to allow the access (i.e., better than threshold $t$) and, if not, what the missing credentials are and what the missing level of cost/trust/preference is. The combination of these weighted procedures leads to a more informative decision, which is necessary, for example, when information is not "crisp".

Notice that other languages can benefit from the underlying semantics provided by $Datalog^W$: we are able to define a weighted version of other trust-management languages based on *Datalog*: for example, Delegation Logic [26], KeyNote [10] and RBAC [34].

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 [23] (i.e., soft security mechanisms).

In the future, we plan to investigate the complexity of tractable soft constraints classes [17] in order to cast them in a tractable *Datalog*-based language. Therefore, we want to extend also the $RT^C$ language [27] (based on *Datalog* enhanced with crisp constraints) in its $RT^{WC}$ soft version.

We want also to study the feasibility of using *Soft Concurrent Constraint Programming*-like languages [2] to represent the same authorization process: the soft constraints, added to the $\sigma$ constraint store by the participating parties, will represent the credentials and the access rules proposed for the request, each statement with its own preference value. In fact, soft constraints [2] extend classical constraints to represent multiple consistency levels, and thus provide a way to express preferences, fuzziness, and uncertainty. Then, a query can be modelled with a simple $ask(c) \longrightarrow^a$ action [2], where the $c$ constraint represents the fact that we want to check if it is implied by the store: the ask succeeds (i.e., the request is accepted) if the store $\sigma$ entails the constraint $c$ and also if the store is "consistent enough" w.r.t. the threshold $a$ on the transition arrow, which represents the minimum requested consistency (or preference) level to be satisfied by the query.

At last, we plan to extend in SWI-Prolog the existent CHR module implementing soft constraints [3], by adding the $\div$ operator inside the soft module. In this way we could implement a stand-alone policy-based authorization system based on constraint programming.

## References


[1] S. Barker and P. J. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. Inf. Syst. Secur.*, 6(4):501–546, 2003.

[2] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.

[3] S. Bistarelli, T. Frühwirth, and M. Marte. Soft constraint propagation and solving in CHRs. In *SAC '02: Proc. of the ACM symposium on Applied computing*, pages 1–5. ACM Press, 2002.

[4] S. Bistarelli and F. Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In *ECAI '06: European Conference on Artificial Intelligence*, pages 63–67, 2006.

[5] S. Bistarelli, F. Martinelli, and F.Santini. A semantic foundation for trust management languages with weights: An application to the rtfamily. In Chunming Rong, Martin Gilje Jaatun, Frode Eika Sandnes, Laurence Tianruo Yang, and Jianhua Ma, editors, *ATC*, volume 5060 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2008.

[6] S. Bistarelli, F. Martinelli, and F.Santini. Deduction and abduction with soft constraints. In *to appear in Autonomic and Trusted Computing, 5th International Conference, ATC 2010*, Lecture Notes in Computer Science. Springer, 2010.

[7] S. Bistarelli, F. Martinelli, and F. Santini. Weighted datalog and levels of trust. In *ARES: Conference on Availability, Reliability and Security*, pages 1128–1134. IEEE Computer Society, 2008.

[8] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[9] S. Bistarelli and F. Rossi. Semiring-based constraint logic programming: syntax and semantics. *ACM Trans. Program. Lang. Syst.*, 23(1):1–29, 2001.

[10] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The keynote trust-management system, rfc2704, 1999.

[11] 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.

[12] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1:146–166, 1989.

[13] 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.

[14] H. Christiansen and V. Dahl. Hyprolog: A new logic programming language with assumptions and abduction. In *Logic Programming, 21st International Conference, ICLP 2005*, volume 3668, pages 159–173. Springer, 2005.

[15] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. M., and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *J. Comput. Secur.*, 9(4):285–322, 2001.

[16] C. Codognet and P. Codognet. Abduction and concurrent logic languages. In *ECAI '94: European Conference on Artificial Intelligence*, pages 75–79. John Wiley and Sons, 1994.

[17] 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.

[18] 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.

[19] 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.

[20] S. N. Foley, W. Mac Adams, and B. O'Sullivan. Aggregating trust using triangular norms in the keynote trust management system. In *Proc. 6th International Workshop on Security and Trust Management (STM 2010), LNCS. Springer*, 2010.

[21] T. W. Frühwirth. *Constraint Handling Rules ; 1 edition (August 10, 2009).* Cambridge Univ. Press, Leiden, 2009.

[22] 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.

[23] 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.

[24] A. C. Kakas. ACLP: Integrating abduction and constraint solving. *CoRR*, cs.AI/0003020, 2000.

[25] H. Koshutanski and F. Massacci. A negotiation scheme for access rights establishment in autonomic communication. *J. Network Syst. Manage.*, 15(1):117–136, 2007.

[26] 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.

[27] 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.

[28] 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*, pages 114–130. IEEE Computer Society, 2002.

[29] M. J. Maher. Abduction of linear arithmetic constraints. In *In ICLP '05: Conf. on Logic Programming*, volume 3668, pages 174–188. Springer, 2005.

[30] M. J. Maher and G. Huang. On computing constraint abduction answers. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 5330 of *LNCS*, pages 421–435. Springer, 2008.

[31] 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.

[32] T. Menzies. Applications of abduction: knowledge-level modelling. *Int. J. Hum.-Comput. Stud.*, 45(3):305–335, 1996.

[33] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. Security policy consistency. *CoRR*, cs.LO/0006045, 2000.

[34] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[35] M. Shanahan. Prediction is deduction but explanation is abduction. In *IJCAI'89: Proc. of the International Joint Conference on Artificial Intelligence*, pages 1055–1060, 1989.

[36] G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *WiSe '04: Workshop of Wireless security*, pages 1–10. ACM, 2004.

[37] E. Thomas and G. Gottlob. The complexity of logic-based abduction. *J. ACM*, 42:3–42, January 1995.

[38] J. Wielemaker. An overview of the SWI-Prolog programming environment. In *Proc. of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium, 2003. Katholieke Universiteit Leuven.