A Formal Framework for Trust Policy Negotiation in Autonomic Systems: Abduction

Stefano Bistarelli, Fabio Martinelli, Francesco Santini

A Formal Framework for Trust Policy Negotiation in Autonomic Systems: Abduction with Soft Constraints*

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

[1] Dipartimento di Matematica e Informatica, Università di Perugia, Italy
[bista,francesco.santini]@dmi.unipg.it
[2] Istituto di Informatica e Telematica (CNR), Pisa, Italy
[stefano.bistarelli,fabio.martinelli,francesco.santini@]iit.cnr.it

Abstract. We show that soft constraints can be used to model logical reasoning, that is deduction and abduction (and induction). In particular, we focus on the abduction process and we show how it can be implemented with a (soft) constraint removal operator. As a running application example throughout the paper, we reason with access control policies and credentials. In this way, we can associate the level of preference defined by the "softness" of the constraint with a "level" of trust. The main benefit comes during the process of automated access authorization based on trust: soft constraint operations can be easily adopted to measure the level of trust required for each operation. Moreover, when the level is not sufficient, abduction can be used to compute the missing credentials and the levels that grant the access, making the request a (weighted) logical consequence. The proposed framework can be used to automate the deduction-abduction negotiation processes.

1 Introduction and Motivations

In this work we interpret *Logical Reasoning* [19, 20] in the soft constraint field [2], in order to grant an authorization based on credentials and trust derivation. The classical operators as deduction and abduction are implemented by soft constraint operators, having in this way a powerful and expressive tool for the automated computation of these related problems, which also take a preference score into account.

Using logic programming languages for security policy is customary in computer security: in [5, 6] the authors have presented a variant of RT language [16] (called it RT^W) that use soft constraints and is able to deal with weights on ground facts and to consequently compute a feedback result for the goal satisfaction. The proposed RT^W framework is able to represent policies: by querying

* This work has been partially supported by EU-FP7-ICT ANIKETOS, EU-FP7-ICT CONNECT projects and MIUR PRIN 20089M932N: "Innovative and multidisciplinary approaches for constraint and preference reasoning".

such programs we can infer if the authorization, given a specific set of credentials, is allowed or not with a specific level of trust. For example, a credential can now state that the referred entity is a "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). Therefore, also the final authorization decision can be taken according to a trust value related to the composition of all the used credentials, e.g. with a total probability greater than 90%. In literature there are many examples where trust or reputation are computed by aggregating some values together [5, 6, 13].

However, often the user simply wants to obtain a service or to access to a resource, and he does not know which credential is needed [15], or which level of trust is needed to obtain the access. Thus, often the user simply presents a very small set of credentials and expects the system to return enough information to guide him towards the missing credentials that grant him to access. When the user obtains such information, he can decide to present such credentials or, for instance, to buy the necessary credentials with "enough" trust level, in order to definitely access to the service. Two basic services can be used inside authorization systems. The first one is represented by deduction [15, 19]: 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 request can be deduced from the policy and the current facts. Abstracting away the details of the policy implementation, we can observe that only one reasoning service is actually used by policy based self-management: deduction. Given a policy, we find out all consequences (actions or obligations) of the policy, i.e. whether granting the request can be deduced from the policy and the current facts.

Access authorization usually needs another reasoning service: abduction [15, 19]. Loosely speaking, abduction is deduction in reverse: given a policy and a request for access to (e.g.) services, it 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. At last we propose also how the third operator, i.e. induction, can be expressed with constraints.

These services provided by logical reasoning are important in autonomic networks of nodes [15], where partners offer services and lightly integrate their efforts into one (hopefully coherent) network. This cross enterprise scenario poses novel security challenges with aspects of trust management systems. Access to services is offered by autonomic nodes on their own and the decision to grant or deny access must rely on attribute credentials sent by the client [15]. There-

fore, when speaking about autonomic authorization, the abduction process it is exactly what one node looks for. By using soft constraints, the user wishes to obtain a resource with a specific trust level, and the server answer has to provide not only the expected credential (that the user needs to present), but also the trust level of such credential. access to a generic student service, sometimes it could be enough to have an id-card, sometimes a personal badge for the student fitness activities, depending on the final trust level.

In Sec. 2 we summarize the fundamental notions about soft constraints, while Sec. 3 shows how classical deduction, abduction and induction operations can be implemented by using constraint composition and division, i.e. $\otimes$ and $\oslash$; to complete the presentation of logical reasoning. Section 4 shows an implementation of deduction/abduction operations by using *Constraint Handling Rules* [12]. At last, Sec. 5 discusses the related work and Sec. 6 draws the final conclusions.

2 Background on Soft Constraints

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 [7]): *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, if S is absorptive, then it is *residuated* if the set $\{x \in A \mid b \times x \leq a\}$ admits a maximum for all elements $a, b \in A$, denoted $a \div b$. With an abuse of notation, the maximal element among solutions is denoted $a \div b$. This choice is not ambiguous: if an absorptive semiring is invertible and residuated, then it is also invertible by residuation, and the two definitions yield the same value.

To use these properties, in [4] it is stated that if we have an absorptive and complete semiring[3], then it is residuated. For this reason, since all classical soft constraint instances (i.e. *Classical*, *Fuzzy*, *Probabilistic* and *Weighted*) are complete and consequently residuated, the notion of semiring division can be

[3] If S is an absorptive semiring, then S is complete if it is closed with respect to infinite sums, and the distributivity law holds also for an infinite number of summands.

applied to all of them. Therefore, for all these semirings it is possible to use the $\div$ operation as a "particular" inverse of $\times$; its extension to soft constraints, defined as $\oslash$, can be used to remove soft constraints from the store.

A *soft constraint* [7, 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.

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 η' is η 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 $(c_1 \otimes c_2)\eta = c_1\eta \times c_2\eta$ (see also [7, 2]). Having defined the operation $\div$ on semirings, 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]. Basing ourselves on the definition of $\div$ and 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 Sec. 3). Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* [7, 2] 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.

Informally, performing the $\otimes$ or the $\oslash$ 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 $c_1 \sqsubseteq c_2 \iff c_1\eta \leq c_2\eta$. Consider the set $\mathcal{C}$ 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 $C \vdash c \iff \bigotimes C \sqsubseteq c$ (see also [2]). Therefore we can say that if, given a constraint store σ, $\sigma \sqsubseteq c$ (or $\sigma \vdash c$), then c is a *logical consequence* of σ.

Considering a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a domain of the variables D, an ordered set of variables V and the corresponding structure $\mathcal{C}$, then $S_C = \langle \mathcal{C}, \otimes, \bar{\mathbf{0}}, \bar{\mathbf{1}}, \exists_x, d_{xy}\rangle$ is a cylindric constraint system (*"a la Saraswat"*)[4].

[4] Notice that in SCCP, algebraicity is not required, since the algebraic nature of $\mathcal{C}$ strictly depends on the properties of the semiring [2].

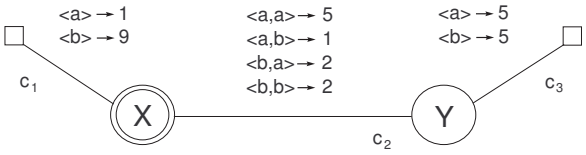

Fig. 1. A soft CSP based on a Weighted semiring.

A SCSP Example. Figure 1 shows a weighted SCSP as a graph: the used semiring is the *Weighted* semiring, i.e. $\langle \mathbb{R}^+, \min, \hat{+}, \infty, 0 \rangle$ ($\hat{+}$ is the arithmetic plus operation). 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. $D = \{a, b\}$. The solution of the CSP in Fig. 1 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 Fig. 1 is 7, related to the solution $X = a$, $Y = b$.

3 Logical Reasoning with Soft Constraints

In this Section we redefine two basic logical reasoning processes [19], deduction and abduction, by using the framework based on soft constraints and presented in Sec. 2. To accomplish this, we adopt the $\otimes$, $\oplus$ and $\sqsubseteq$ operators as summarized in Sec. 2. We show that both deduction and abduction processes can be described them.

Deduction. Deduction means determining the conclusion, thus to use a rule and its precondition to make a conclusion [15, 19]. An Example is "When a paper is poorly presented, it receives a bad review. The paper I am reviewing is poorly written, then I write a bad review". An argument is said to be deductive when the truth of the conclusion necessarily follows or is a logical consequence of the truth of the premises and (consequently) its corresponding conditional is a necessary truth. Deductive arguments are said to be valid or invalid, never true or false. A deductive argument is valid if and only if the truth of the conclusion actually does follow necessarily (or is indeed a logical consequence of) the premises and (consequently) its corresponding conditional is a necessary truth. In Def. 1 we define deduction through soft constraints:

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

Therefore, w.r.t. Def. 1, the store σ represents the premise and c a logical consequence of σ. The $\vdash$ (or $\sqsubseteq$, see Sec. 2) consequently implements the deduction operator as for the crisp $\vdash$ entailment operator [10].

As introduced in Sec. 1, we use a running example based on Policy-based Access Control to show how "soft" deduction and abduction can be used. We suppose that the soft constraint store contains all the collected information, i.e. the access policy and the credentials presented by the requestor and other parties, that is, for example, external databases. 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. With respect to Def. 1, the σ store represents the policy p and the collected credentials c, i.e. $\sigma = p \otimes c$, and the store entails (i.e. deduces) the access request r if $\sigma \sqsubseteq r$.

An example of policy and credentials is given in the program in Tab. 1: we adopt the RT_0^W language [5, 6], a weighted extension of the well-known RT_0 language [16].

```
EPub.disct ⟵ EPub.preferred ∩ EPub.brightStudent.
EPub.preferred ⟵ EOrg.highBudget ∩ EOrg.oldCustomer.
EPub.brightStudent ⟵ EPub.goodUniversity.highMarks.
EPub.goodUniversity ⟵ ABU.accredited.
ABU.accredited ⟵ ⟨ StateU, 9 ⟩.
StateU.highMarks ⟵ ⟨ Alice, 8 ⟩.
EOrg.highBudget ⟵ ⟨ Alice, 6 ⟩.
EOrg.oldCustomer ⟵ ⟨ Alice, 7 ⟩.
```

Table 1. An example in RT_0^W, with weights associated to the credentials.

The example in Tab. 1 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 cost 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. To solve the example in Tab. 1, we use a *Weighted* semiring $S_{Weighted} = \langle \mathbb{R}^+, min, \hat{+}, +\infty, 0 \rangle$, where $\hat{+}$ is the arithmetic sum and where the preference levels in $\mathbb{R}^+$ represents the money cost that is needed to buy or re-

trieve a given credential (e.g. the cost to pay the office supplies and the clerk service): for example, `StateU.highMarks` $\longleftarrow$ $\langle$ `Alice, 8` $\rangle$ in Tab. 1 certifies that Alice has obtained a good number of high marks for the exams completed at the StateU university (the credential is issued by StateU), and the cost associated with this credential is 8 euro.

The last four rules represent the weighted credentials presented to the authorization entity, while the other rules consist in the access policy. The example, together with the deduction process, is described in the next paragraph. The final money cost, 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 7 are authorized. The four credentials, which are all used by the policy rules, prove that Alice is eligible for the discount with a money cost of 30: $6\hat{+}7\hat{+}8\hat{+}9 = 30$ euro.

By considering Def. 1 and the example in Tab. 1, we can use the deduction operation in the following simple and crisp example (also classical crisp problems can be cast in the soft framework [7, 2] by using the *Classical* semiring $\langle\{true, false\}, \vee, \wedge, false, true\rangle$): if σ = *Student(Alice)* $\otimes$ *HighMarks(Alice)* $\otimes$ *(Student(x)*$\wedge$*HighMarks(x)* $\sqsubseteq$ *Access(x))*, then $\sigma \sqsubseteq$ *Access(Alice)*, we can deduce that Alice can access. The policy in Tab. 1 has been also implemented in *CIAO Prolog* in [6].

Abduction. Abduction consists in using the conclusion and the rule to support that the precondition could explain the conclusion [15, 19]. For example, "When a paper is poorly presented, it receives a bad review. I received a bad review for my paper, therefore it is poorly presented". Therefore, if the σ 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 Def. 2 we define abduction by using the $\oslash$ operation presented in Sec. 2:

Definition 2. [Abduction with soft constraints] *Given a soft constraint store σ and a constraint c, then the abduction process is aimed at finding a constraint d, such that $\sigma \otimes d \sqsubseteq c$, i.e. $d = c \oslash \sigma$.*

Notice that the division operator we are adopting is a deterministic operator (see Sec. 2) and provides a single solution d (which is also the minimal solution, as explained in the following of this section). Considering Def. 2, the trivial case is when $\sigma \sqsubseteq c$ since $c \oslash \sigma = \bar{\mathbf{1}}$, that is nothing must be added to the store in order to satisfy c, which is already implied by it.

Access Control for autonomic communication needs the abduction reasoning service [22]. 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. If we consider our application example on policy-based access control, $\sigma = p \otimes c$ represent the policy p and a c credential, and the abducted d constraint is such that $\sigma \otimes d \sqsubseteq r$, i.e. $d = r \oslash \sigma$, where d is the abducting 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 [10]. 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 Def. 2 is able to find the minimal (in this case, soft) explanation as proposed in [10] for the crisp version, according to the definition of $\div$, and consequently $\oplus$ shown in Sec. 2 and in [4]. Therefore, the d constraint in Def. 2 is the minimal explanation of c.

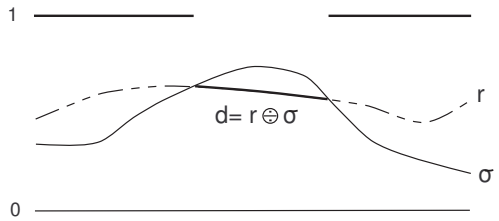

Fig. 2. The representation of σ, the request r and the abducted soft constraint d, represented with the thick line.

A graphical intuition of the "soft" abduction definition is given in Fig. 2. There we consider the *Fuzzy* semiring $S_{Fuzzy} = \langle [0..1], max, min, 0, 1 \rangle$, that is we analyze the truth degree connected to a credential and evaluated by the entity which signs and issues it, as proposed for the examples in [5, 6]. Suppose that the soft constraints σ and r (and consequently $d = r \oplus \sigma$) have as their support only the variable x, which represents the entity that requests the authorization; these constraints are represented as functions in x in Fig. 2. Suppose that σ = *Student(Alice)* $\otimes$ *(Student(x)* $\wedge$ *HighMarks(x)* $\sqsubseteq$ *Access(x))*, and that r = *Access(x)*, then d = *HighMarks(x)*. Moreover, if σ*(x= Alice)* = 0.9 and r*(x= Alice)* = 0.7, then d*(x= Alice)* = 0.7. Thus Alice needs to attest the fact that she has high marks with a trust score 0.7 or greater. Where $\sigma(x) \leq r(x)$, then $d(x) = \mathbf{1}$, 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 *Weighted* semiring $S_{Weighted} = \langle \mathbb{R}^+, min, \hat{+}, +\infty, 0 \rangle$. If σ*(x= Alice)* = 7 and r*(x= Alice)* = 10, then d*(x= Alice)* = 3 ($d = r \oplus \sigma$). Since d = *HighMarks(x)*, this operation suggests Alice to provide a credential attesting that she has $HighMarks$ and to spend at least 3 euro, in order to be

able to access. The definition of $\div$ (i.e. the arithmetic subtraction $\hat{-}$) is:

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

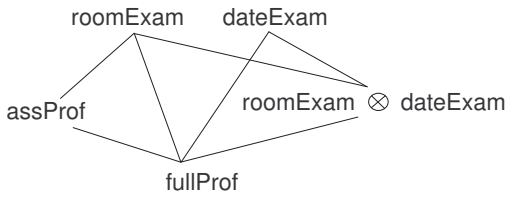

Fig. 3. A lattice representing the strength of roles: *fullProf* entails all the other roles.

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 [15]. 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, *fullProf(X)* $\sqsubseteq$ *dateExam(X)* $\otimes$ *roomExam(X)*, but *assProf(X)* $\sqsubseteq$ *roomExam(X)* only, i.e. the assistant professor has less rights. The complete lattice of roles is shown in Fig. 3.

It is worth to notice that our $\oplus$ operation can be considered as a "relaxation" of the store, and not only as a strict removal of the token representing the constraint, because in soft constraints we do not have the concept of token. Thus, if we perform $\sigma \oplus c$ during the abduction phase, c can be removed even if it is different from any other constraints previously added to σ; therefore, it is also possible to remove only a part of a piece of information (e.g. a part of a credential). For example, considering the five soft constraints reported in Fig. 4 and the *Weighted* semiring, $c_1 \otimes c_2 = c_3$, and at this point we can also remove c_4 from c_3, that is $c_3 \oplus c_4 = c_5$, even if c_4 has been not already added to the store before, but it is however implied by it. To show this concept with a crisp example, it is like if the constraint store is equivalent to *I have a car* and we could remove *A tire of my car is punctured* obtaining *I have a car with three not punctured tyres*.

Induction. Logical reasoning includes also a third kind of reasoning, i.e. *induction*, also known as *inductive reasoning* or *inductive logic*, that is a type of reasoning which involves moving from a set of specific facts to a general conclusion. In words, induction means determining the "rule" that leads to the conclusion

$$c_1 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_1(x) = x + 3 \qquad c_2 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_2(x) = x + 5$$
$$c_3 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_3(x) = 2x + 8 \qquad c_4 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_4(x) = x + 1$$
$$c_5 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_5(x) = x + 7$$

Fig. 4. Five weighted soft constraints; notice that $c_3 = c_1 \otimes c_2$ and $c_5 = c_3 \oplus c_4$.

from precondition. An example of induction could start from the fact that "I have always received bad reviews when writing papers about bioinformatics", it is possible to learn the rule "If I write papers about bioinformatics, then I will receive a bad review" (because, for example, my interests cover only trust and security).

Induction is a valuable tool for autonomic nodes, because complete and consistent access policies may be difficult to write [15]. For example, if a node has only a partial policy and an additional set of examples of access, both permitted or forbidden. This node should be able, by generalizing from the examples, to derive a policy that matched the given examples and is also able to answer other similar queries.

Even if this paper is mainly focused on the abduction process, we can hint that constraint have been already used in literature to solve also this problem: in works as [8, 23] the authors suggest learning algorithms that deduce (crisp or soft) constraints from assignments of values to sets of variables. This can be used when the formulation of the problem is not available, or also to find new constraints in order to fasten the search of the solution, by further reducing the allowed assignment of variables.

In [1] the authors simulate the task of learning an SCSP, given a fixed constraint graph and some examples of solution ratings. It is assumed that the level of preference for some solutions is known, and a suitable learning technique is defined to learn the values to be associated with each constraint tuple, in a way that is compatible with the level of preference.

4 Deduction and Abduction in CHR

In this section we implement the deduction and abduction operations presented in Sec. 3 by using the *Constraint Handling Rules* [12] (CHR) language.

CHR 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* [24].

```
:- 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, 9), highMarks(stateU, alice, 8),
     highBudget(eOrg, alice, 6), oldCustomer(eOrg, alice, 7),
     discount(alice, 25).

access(alice)

2 ?- accredited(aBU, stateU, 9), highMarks(stateU, alice, 8),
     highBudget(eOrg, alice, 6), discount(alice, 25).

brightStudent(alice, 17)
highBudget(eOrg, alice, 6)
discount(alice, 25)
```

Fig. 5. Deduction with CHR rules: solving the example in Tab. 1.

In Fig. 5 we implement the deduction process by translating the RT_0^W program in Tab. 1. Given the five simplification rules (i.e., $policy1, \ldots, policy4$), which represent the access policy, and the first query, (i.e. 1 ?-), that is *accredited(aBU, stateU, 9), highMarks(stateU, alice, 8), highBudget(eOrg, alice, 6), oldCustomer(eOrg, alice, 7), discount(alice, 25).*) which represents the set of presented credentials and the discount request (i.e. *discount(alice, 25)*), we are able to deduce that Alice can have discount. In fact the system answers with *access(alice)*. The solver can fire the first rule (i.e. *policyrule1*) because the sum of the trust level of the credentials, which is = 30 is greater than the requested threshold in the discount request, which is 25. As a remind, in this example (as for the one in Tab. 1) we use the weighted semiring, that is the $\times$ operator of the semiring is modeled with the arithmetic addition.

The second query (i.e. 2 ?-) in Fig. 5 is presented to show what happens when any of the credentials is not presented to the authorization system: without the *oldCustomer(eOrg, alice, 7)* credential missing in the query, the CHR rules in Fig. 5 are not able to compute the consequences in *policyrule2* rule, that is Alice is *preferred* for *eOrg*.

In this case, that is when the *access* constraint is not added to the store and, therefore, the access is not granted, the rules in Fig. 6 can be used to abduce the missing credentials with the (minimum) required level of trust. The query is

```
:- 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, 17), highBudget(eOrg, alice, 6), discount(alice, 25).

oldCustomer(eOrg, alice, 2)

2 ?- discount(alice, 25).

highBudget(_G88354, alice, _G88403)
oldCustomer(_G88354, alice, _G88356)
comb(25)
```

Fig. 6. Abduction with CHR rules.

now represented by the store obtained at the end of the failed access in Fig. 5, i.e. *brightStudent(alice, 17), highBudget(eOrg, alice, 6), discount(alice, 25)*. The program answers that we need the *oldCustomer* credential with a level of trust equal or greater than 2: the sum of the weights of the other two credentials is 23 and the discount is required with 25. Notice that, w.r.t. the policy in Tab. 1, in Fig. 6 we show only a subset of the abduction rules for sakes of brevity, i.e. the subset concerning the branch under *EPub.preferred*. This subset represent only the rules needed to answer to the first abduction query, since the *brightStudent* has been already deduced by the program in Fig. 5: the other (not shown here) abduction rules are not used in this example.

In the second query in Fig. 6 the program abduce both the missing credentials (i.e. *oldCustomer* and *highBudget*) needed to receive the discount. Moreover, it replays with a further constraint stating that the combination (i.e. the $\times$ semiring operator) of trust scores for these credentials must be 25 at least (i.e. *comb(25)*).

The deduction/abuction processes in CHR can be adopted to implement trust negotiation, as represented in the system architecture in Fig. 7: *1)* the client sends its credentials in order to be authorized, *2)* the server uses its CHR solver to deduce the authorization or to abduce the missing credentials, that *3)* are sent back to the client; *4)* the client then uses its solver to decide, for example, the appropriate level of trust to be assigned to each missing credential, in order to reach (by combining them) the required threshold (i.e. 25 for the example in Fig. 7). In this way, the negotiation process is interactive.

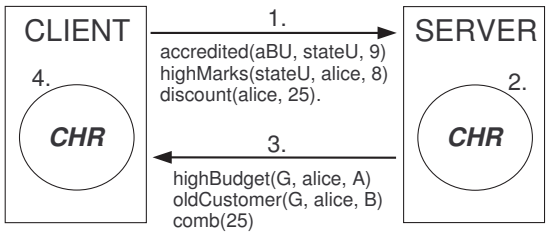

Fig. 7. Trust negotiation with CHR solvers.

5 Related Work

The importance of the deduction and abduction operations for policy-based management of autonomic networks has been already highlighted in [15]. An Autonomic Network 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 [15] 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. [17] where he investigates abduction applied to fully-defined predicates, specifically linear arithmetic constraints over the real numbers. In [18], 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 [21] the authors present a CHR-based tool for detecting security policy inconsistencies, where policy are represented by CHR rules.

In [11] the authors consider configuration problems with preferences rather than just hard constraints, and we analyze and discuss the features that such preference-based configurators should have. In particular, these configurators, should provide explanations for the current state, implications of a future choice. is done by keeping track of the inferences that are made during the constraint propagation enforcing phases.

Concerning instead systems to implement abduction and deduction processes, abduction reasoning has been already realized in *HYPROLOG* [9] (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. Another implementation of this kind of system is represented by ACLP [14]; 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.

6 Conclusions

In this paper we have proposed how two important logical reasoning processes, i.e. deduction and abduction can be modeled and solved with soft constraints [2]. We mainly focused on the abduction phase, since deduction was already available in the framework with the classical (soft) constraint entailment. Abduction can be implemented through the use of the $\oplus$ operator (see Sec. 2).

Deduction and abduction are presented with the help of a running example based on policy-based access control, in order to immediately describe a possible application of these operators and their importance to derive knowledge. We manage trust-based access authorization with weighted credentials, i.e. credentials where the associated level of trust can be used to enrich the authorization process. The proposed framework can be used to improve logical reasoning processes in autonomic networks of nodes [15], since these services (i.e. deduction and abduction) help the self-managing characteristics of distributed computing resources: a node sometimes needs to independently produce the missing information (through an abduction process).

As a future work, 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. The distributed negotiation architecture shown in Fig. 7 can be enriched in order to model the mutual disclosure of credentials between strangers when credentials are sensitive.

References

1. A. Biso, F. Rossi, and A. Sperduti. Experimental results on learning soft constraints. In *KR*, pages 435–444, 2000.
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 RT family. In *Autonomic and Trusted Computing, 5th International Conference*, volume 5060 of *LNCS*, pages 481–495. Springer, 2008.
6. 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.

7. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
8. C.Bessière, R. Coletta, and T. Petit. Learning implied global constraints. In *IJCAI'07: Proc. of the International Joint Conference on Artificial Intelligence*, pages 44–49, 2007.
9. 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.
10. 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.
11. E. C. Freuder, C. Likitvivatanavong, M. Moretti, F. Rossi, and R. J. Wallace. Computing explanations and implications in preference-based configurators. In *International Workshop on Constraint Solving and CLP*, volume 2627 of *LNCS*, pages 76–92. Springer, 2002.
12. T. W. Frühwirth. Constraint handling rules. In *Selected Papers from Constraint Programming*, pages 90–107, London, UK, 1995. Springer-Verlag.
13. 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.
14. A. C. Kakas. ACLP: Integrating abduction and constraint solving. *CoRR*, cs.AI/0003020, 2000.
15. H. Koshutanski and F. Massacci. A negotiation scheme for access rights establishment in autonomic communication. *J. Network Syst. Manage.*, 15(1):117–136, 2007.
16. 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.
17. M. J. Maher. Abduction of linear arithmetic constraints. In *In ICLP '05: Conf. on Logic Programming*, volume 3668, pages 174–188. Springer, 2005.
18. 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.
19. T. Menzies. Applications of abduction: knowledge-level modelling. *Int. J. Hum.-Comput. Stud.*, 45(3):305–335, 1996.
20. H. E. Pople. On the mechanization of abductive logic. In *IJCAI'73: Proc. of the International Joint Conference on Artificial Intelligence*, pages 147–152, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.
21. C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. Security policy consistency. *CoRR*, cs.LO/0006045, 2000.
22. 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.
23. X. Vu and B. O'Sullivan. Semiring-based constraint acquisition. *Tools with Artificial Intelligence, IEEE International Conference on*, 1:251–258, 2007.
24. 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.