Inference Leakage Detection for Authorization Policies over RDF Data

Tarek Sayah(✉), Emmanuel Coquery, Romuald Thion, and Mohand-Saïd Hacid

Université de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, 69622 Lyon, France
{tarek.sayah,emmanuel.coquery,romuald.thion, mohand-said.hacid}@liris.cnrs.fr

Abstract. The Semantic Web technologies include entailment regimes that produce new RDF data from existing ones. In the presence of access control, once a user has legitimately received the answer of a query, she/he can derive new data entailed from the answer that should have been forbidden if carried out inside of the RDF store. In this paper, we define a fine-grained authorization model for which it is possible to check in advance whether such a problem will arise. To this end, we provide a static analysis algorithm which can be used at the time of writing the authorization policy and does not require access to the data. We illustrate the expressiveness of the access control model with several conflict resolution strategies including *most specific takes precedence* as well as the applicability of the algorithm for diagnosis purposes.

Keywords: Authorization · Semantic reasoning · Inference leakage

1 Introduction

According to World Wide Web Consortium (W3C), inference on the Semantic Web using the Resource Description Framework (RDF) "*improve the quality of data integration on the Web, by discovering new relationships, automatically analyzing the content of the data*". Inference rules are used to derive new triples from those explicitly asserted in a RDF store. In particular, a set of inference rules known as RDF *Schema* (RDFS) is standardized [6]. Authorization models for RDF data have been proposed to control accesses to RDF data, both in the presence of inference rules [7,8,10,15] or not [1,5,13]. However, the issue is that inference capabilities can be used by a malicious user to infer sensitive information from public ones. We call this problem the *inference leakage* problem.

To illustrate the so-called inference leakage problem, suppose that RDF triples stating that someone has a cancer are labeled as confidential (*e.g.*, triples similar to (?p ; `rdf:type` ; `:cancerous`) with ?p denoting a person), while the ones stating that a person has a tumor are public (*e.g.*, triples of the form (?p; `:hasTumor`; ?t)). If there exists a public triple stating that the domain of the

P. Samarati (Ed.): DBSec 2015, LNCS 9149, pp. 346–361, 2015.
DOI: 10.1007/978-3-319-20810-7_24

:hasTumor predicate is :cancerous (*e.g.*, (:hasTumor ;rdfs :dom ;:cancerous)) then, using the RDFS rule that relates the domain of a predicate to the type of its subjects, sensitive information can be inferred from the authorized triples. The situation is even worse when RDFS is enriched with user-defined rules.

The issue is that such inferences can be performed outside the RDF store, using only authorized data. One way of preventing inference leakages could be to dynamically deny queries that may provide too much information, at the price of a (possibly) quite high runtime overhead. In this paper, we propose an alternative approach based on a static analysis. The idea is to detect, at the time of specifying the confidentiality policy, whether authorizations and inference rules interact in such a way they can lead to disclose sensitive information. Several authorization models for RDF which consider inference use annotations to determine whether the inferred triples are accessible or not [8,10,15]. The problem is that these approaches do not guarantee that forbidden information cannot be inferred again, once the data have been disclosed. The inference leakage problem in the case of RDFS has been investigated by Jain and Farkas [7], but the base RDF graph kept in the RDF store is needed and conflict resolution strategies are hard-coded in their algorithm. Related works are discussed in Sect. 6.

We highlight the main contributions of this paper and detail its organization. First of all, by using standard machinery for RDF query and entailment defined in Sect. 2, we propose a flexible access control framework for RDF data in Sect. 3. The access control semantics is defined by computing the *authorized subgraph* G^+ of a base RDF graph G, and hence it is independent of the query language used by the RDF store. In Sect. 3.2, we identify and formalize a *consistency property* that captures the information leakage arising when inference rules and authorizations interact, as exemplified informally in this introduction. Intuitively, a policy is consistent w.r.t. a set of inference rules $\mathbf{R}$ if the authorized subgraph G^+ of a closed graph G is itself *closed*, that is, no new facts can be produced using $\mathbf{R}$ another time. In Sect. 4, we illustrate the applicability of the authorization model by showing that usual conflict resolution strategies can be expressed in our framework. In particular, we show that the *most specific takes precedence* strategy can be modeled, this strategy being particularly useful to capture exceptions in authorizations. In Sect. 5, we propose an algorithm that, given a policy P and a set of inference rules $\mathbf{R}$, but without any prior knowledge of G, checks if the consistency property holds. The algorithm is proved correct[1] and it is constructive: whenever the answer is positive, a counterexample graph is computed. This answer can be used by the administrator to analyze and then solve the issue, as illustrated in Sect. 5.2. We conclude and discuss ongoing and future work in Sect. 7.

2 Data Model

2.1 RDF and SPARQL

RDF is a generic, graph-oriented data model that represents information based on triples of the form "(subject ;predicate ;object)" built from pairwise

[1] Proofs are provided at http://liris.cnrs.fr/~tsayah/DBSEC2015/.

disjoint countably infinite sets I, V, and L for IRIs, variables, and literals respectively. A set of RDF triples is called an RDF graph. RDF graphs are stored into repositories usually called RDF stores. In this paper, we reuse the formal definitions and notation used by Pérez and Gutierrez [11]. Throughout the paper, $\mathcal{P}(\mathsf{E})$ denotes the *finite powerset* of a set E and $\mathsf{F} \subseteq \mathsf{E}$ denotes a *finite subset* F.

We do not explicitly use blank nodes which are replaced by *variables*. Blank nodes of RDF are semantically equivalent to existentially quantified variables [12]. Not to distinguish between blank nodes and variables significantly reduces the overhead of formal definitions but it does not change the expressiveness of the framework.

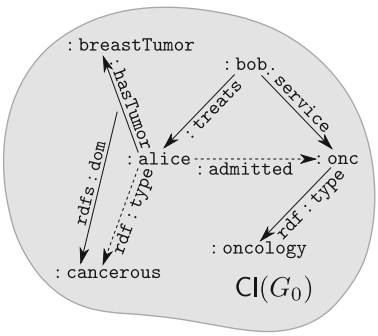
RDF graphs are queried using SPARQL which is the RDF counterpart of the SQL query language used in relational databases. We focus on a subset of SPARQL called *basic graph patterns* used in Sect. 3 to define authorizations and policies.

Definition 1 (Triple Pattern, Graph Pattern). *A* term $\mathtt{t}$ *is either an IRI, a variable or a literal. Formally* $\mathtt{t} \in \mathsf{T} = \mathsf{I} \cup \mathsf{V} \cup \mathsf{L}$ *. A tuple* $t \in \mathsf{TP} = \mathsf{T} \times \mathsf{T} \times \mathsf{T}$ *is called a* Triple Pattern (TP)*. A* Basic Graph Pattern (BGP)*, or simply a* graph*, is a finite set of triple patterns. Formally, the set of all BGPs is* $\mathsf{BGP} = \mathcal{P}(\mathsf{TP})$*.*

Given a triple pattern $tp \in \mathsf{TP}$*,* $\mathsf{var}(tp)$ *is the set of variables occurring in* tp*. Similarly, given a basic graph pattern* $B \in \mathsf{BGP}$*,* $\mathsf{var}(B)$ *is the set of variables occurring in the BGP defined by* $\mathsf{var}(B) = \{v \mid \exists tp \in B \wedge v \in \mathsf{var}(tp)\}$*.*

When graph patterns are considered as instances stored in an RDF store, we simply call them *graphs*. In this paper, we do not make any formal difference between a basic graph pattern and a graph. Also, note that Definition 1 is slightly more liberal than usual because variables are allowed in property positions.

Example 1. Figure 1 depicts a graph G_0 constituted by triples et_1 to et_5, both pictorially and textually. We explicitly write `rdf` and `rdfs` when the term is from the RDF or the RDFS standard vocabulary. However, we do not prefix the other terms for the sake of simplicity. Triples it_1 an it_2 are depicted by dashed arrow in Fig. 1. They are part of the closure $\mathsf{Cl}(G_0)$ of G_0 that we will introduce in Sect. 2.2.



	Subject	Predicate	Object
et_1	:hasTumor	rdfs:dom	:cancerous
et_2	:onc	rdf:type	:oncology
et_3	:alice	:hasTumor	:breastTumor
et_4	:bob	:service	:onc
et_5	:bob	:treats	:alice
it_1	:alice	rdf:type	:cancerous
it_2	:alice	:admitted	:onc

Fig. 1. An example of an RDF graph G_0 and its closure $\mathsf{Cl}(G_0)$

The evaluation of a graph pattern B on another graph pattern G is given by mapping the variables of B to the terms of G such that the structure of B is preserved. First, we define the substitution mappings as usual. Then, we define the evaluation of B on G as the set of substitutions that embed B into G.

Definition 2 (Substitution Mappings). *A* substitution (mapping) η *is a partial function* $\eta : \mathsf{V} \rightarrow \mathsf{T}$. *The domain of* η, $\mathsf{dom}(\eta)$, *is the subset of* V *where* η *is defined. We overload notation and also write* η *for the partial function* $\eta^\star : \mathsf{T} \rightarrow \mathsf{T}$ *that extends* η *with the identity on terms. Given two substitutions* η_1 *and* η_2, *we write* $\eta = \eta_1\eta_2$ *for the substitution* $\eta : ?\mathtt{v} \mapsto \eta_2(\eta_1(?\mathtt{v}))$.

Given a triple pattern $tp = (\mathtt{s}\,;\mathtt{p}\,;\mathtt{o}) \in \mathsf{TP}$ *and a substitution* η *such that* $\mathsf{var}(tp) \subseteq \mathsf{dom}(\eta)$, $(tp)\eta$ *is defined as* $(\eta(\mathtt{s})\,;\eta(\mathtt{p})\,;\eta(\mathtt{o}))$. *Similarly, given a graph pattern* $B \in \mathsf{BGP}$ *and a substitution* η *such that* $\mathsf{var}(B) \subseteq \mathsf{dom}(\eta)$, *we extend* η *to graph pattern by defining* $(B)\eta = \{(tp)\eta \mid tp \in B\}$.

Definition 3 (BGP Evaluation). *Let* $G \in \mathsf{BGP}$ *be a graph, and* $B \in \mathsf{BGP}$ *a graph pattern. The evaluation of* B *over* G *denoted by* $[\![B]\!]_G$ *is defined as the following set of substitution mappings:*

$$[\![B]\!]_G = \{\eta : V \rightarrow T \;\mid\; \mathsf{dom}(\eta) = \mathsf{var}(B) \wedge (B)\eta \subseteq G\}$$

Example 2. Let B be defined as $B = \{(?\mathtt{d}\,;\mathtt{:service}\,;?\mathtt{s}), (?\mathtt{d}\,;\mathtt{:treats}\,;?\mathtt{p})\}$. The evaluation of B on the example graph G_0 of Fig. 1 is $[\![B]\!]_{G_0} = \{\eta\}$, where η is defined as $\eta : ?\mathtt{d} \mapsto \mathtt{:bob}$, $?\mathtt{s} \mapsto \mathtt{:onc}$ and $?\mathtt{p} \mapsto \mathtt{:alice}$.

Formally, the definition of BGP evaluation captures the semantics of SPARQL restricted to the conjunctive fragment of `SELECT` queries that do not use `FILTER`, `OPT` and `UNION` operators (see [11] for further details). Please note that this fragment is basically used to define the access control model itself, and it is not meant to replace the generic SPARQL query language on RDF stores.

2.2 Inference Rules and RDFS

Inference rules are used to add triples to a graph when it contains triples conforming to a graph pattern. Thus, inference rules turn an RDF store into a *deductive* database similar to positive Datalog that extends traditional (non-deductive) relational databases.

Definition 4 (Inference Rule). *An* inference rule $\mathbf{r}$ *is a formal expression of the form* $(tp \leftarrow tp_1, \ldots, tp_k)$ *where* $tp, tp_0, \ldots, tp_k \in \mathsf{TP}$ *that is subjected to the condition* $\mathsf{var}(tp) \subseteq \mathsf{var}(\{tp_0, \ldots, tp_k\})$. *The sets of inference rules are denoted by* $\mathbf{R}$.

For a rule $(tp \leftarrow tp_1, \ldots, tp_k)$, the condition $\mathsf{var}(tp) \subseteq \mathsf{var}(\{tp_0, \ldots, tp_k\})$ ensures that it does not introduce fresh uninstantiated variables when applied to a graph. When useful, we also use the notation $\frac{tp_1,\ldots,tp_k}{tp}$ for inference rules. We define an operational semantics for the rules, inspired by the *fixpoint semantics* of Datalog. It is known that the closure of a finite graph is finite and the operator is increasing, monotonic and idempotent [2, Chap. 12].

Definition 5 (Rule Semantics, Closure). *Given a graph pattern* $G \in \mathsf{BGP}$ *and an inference rule* $\mathbf{r} = (tp \leftarrow tp_1, \ldots, tp_k)$*, the* set of triples (immediately) deduced from G by $\mathbf{r}$ *is* $\phi_{\mathbf{r}}(G) = \{(tp)\sigma \mid \sigma \in [\![\{tp_1, \ldots, tp_k\}]\!]_G\}$*. We extend the operator* $\phi(G)$ *to sets of inference rules* $\mathbf{R}$*,* $\phi_{\mathbf{R}}(G) = \bigcup_{\mathbf{r}\in\mathbf{R}} \phi_{\mathbf{r}}(G)$*.*

Given a set of inference rules $\mathbf{R}$*, let* $(G_i)_{i\in\mathbb{N}}$ *be the infinite sequence of basic graph patterns defined by* $G_0 = G$ *and for any* $i \in \mathbb{N}$*,* $G_{i+1} = G_i \cup \phi_{\mathbf{R}}(G_i)$*. The* closure *of* G *w.r.t.* $\mathbf{R}$ *is* $\mathsf{Cl}_{\mathbf{R}}(G) = \bigcup_{i\in\mathbb{N}} G_i$*. We write* $\mathsf{Cl}(G)$ *when* $\mathbf{R}$ *is clear from the context. We say that a graph is* closed *when* $\mathsf{Cl}_{\mathbf{R}}(G) = G$

Example 3. The following RDFS rule named **RDom** states that the type of a triple's subject is the class defined by its predicate's domain. Let us consider the graph G_0 of Fig. 1. If we apply the inference rule **RDom** using triples et_1 and et_3 then we infer it_1. Thus, $\mathsf{Cl}_{\{\mathbf{RDom}\}}(G_0) = G_0 \cup \{it_1\}$.

$$\frac{(?\mathtt{p}\,;\mathtt{rdfs:dom}\,;?\mathtt{d})(?\mathtt{x}\,;?\mathtt{p}\,;?\mathtt{y})}{(?\mathtt{x}\,;\mathtt{rdfs:type}\,;?\mathtt{d})} = \mathbf{RDom}$$

Assume that we add an extra rule name **RAdm** which states that if a doctor is assigned to a service and treats a patient, then this patient is admitted to the doctor's service. Referring to the graph G_0 of Fig. 1, its closure now contains a new inferred triple $\mathsf{Cl}_{\{\mathbf{RDom},\mathbf{RAdm}\}}(G_0) = G_0 \cup \{it_1, it_2\}$.

$$\frac{(?\mathtt{d}\,;\mathtt{:service}\,;?\mathtt{s})(?\mathtt{d}\,;\mathtt{:treats}\,;?\mathtt{p})}{(?\mathtt{p}\,;\mathtt{:admitted}\,;?\mathtt{s})} = \mathbf{RAdm}$$

3 Access Control

In this section, we define an access control model for RDF that uses the ingredients from Sect. 2 and we formalize a consistency property between authorizations and inference rules that captures the absence of information leakage.

We assume that the Policy Decision Point (PDP) knows what are the authorizations applicable to a given authenticated requester. The entity to which authorizations are granted or denied is left implicit in this paper. The upstream mapping from requesters to authorizations may use any model from the literature, for instance using users' identifiers, groups, roles or set of attributes. In other words, we assume that the PDP is able to produce a set of authorizations in our formalism for each requester. Moreover, we restrict ourselves to the read action on RDF graphs, because the information leakage issue in the presence of RDF inference already exists in this minimal setting. The investigations on upstream policy definitions, their administration as well as update, delete and insert actions are left for future work.

3.1 Authorization Policy

We define authorizations using basic SPARQL constructions, namely basic graph patterns, in order to facilitate the administration of access control and to include homogeneously authorizations into concrete RDF stores with minimal effort.

Definition 6 (Authorization). *Let* $\mathsf{Eff} = \{+, \text{-}\}$ *be the set of applicable* effects. *Formally, an* authorization $\mathfrak{a} = (e, h, b)$ *is a element of* $\mathsf{Auth} = \mathsf{Eff} \times \mathsf{TP} \times \mathsf{BGP}$. *The component* e *is called* the effect *of the authorization* $\mathfrak{a}$, h *and* b *are called its* head *and* body *respectively. We use the function* $\mathsf{effect} : \mathsf{Auth} \to \mathsf{Eff}$ *(resp.,* $\mathsf{head} : \mathsf{Auth} \to \mathsf{TP}$, $\mathsf{body} : \mathsf{Auth} \to \mathsf{BGP}$*) to denote the first (resp., second, third) projection function. We call* $\mathsf{hb}(\mathfrak{a}) = \{\mathsf{head}(\mathfrak{a})\} \cup \mathsf{body}(\mathfrak{a})$ *the* underlying graph pattern *of the authorization* $\mathfrak{a}$. *Given a finite set of authorizations* $\mathfrak{A}$, *we introduce* $\mathfrak{A}^+ = \{\mathfrak{a} \in \mathfrak{A} \mid \mathsf{effect}(\mathfrak{a}) = +\}$ *and* $\mathfrak{A}^- = \{\mathfrak{a} \in \mathfrak{A} \mid \mathsf{effect}(\mathfrak{a}) = \text{-}\}$ *for the positive and negative subsets of* $\mathfrak{A}$.

We use the concrete syntax "`GRANT/DENY` h `WHERE` b" to represent an authorization $\mathfrak{a} = (e, h, b)$. We use the `GRANT` keyword when $e = +$ and the `DENY` keyword when $e = \text{-}$. Condition `WHERE` $\emptyset$ is elided when b is empty.

Example 4. Consider the set of authorizations shown in Table. 1. Authorization $\mathfrak{a}_1$ grants access to triples with predicate :`hasTumor`. Authorization $\mathfrak{a}_5$ states that triples about admission to the oncology service are specifically denied, whereas the authorization $\mathfrak{a}_6$ states that such information are allowed in the general case. Finally, authorization $\mathfrak{a}_9$ denies access to any triple, it is meant to be a default authorization.

Given an authorization $\mathfrak{a} \in \mathsf{Auth}$ and a graph G, we say that $\mathfrak{a}$ is *applicable on* a triple $t \in G$ if there exists a substitution θ such that the head of $\mathfrak{a}$ is mapped to t and all the conditions expressed in the body of $\mathfrak{a}$ are satisfied as well. In other words, we evaluate the underlying graph pattern $\mathsf{hb}(\mathfrak{a}) = \{\mathsf{head}(\mathfrak{a})\} \cup \mathsf{body}(\mathfrak{a})$ against G and we apply all the answers of $[\![\mathsf{hb}(\mathfrak{a})]\!]_G$ to $\mathsf{head}(\mathfrak{a})$. In a concrete system, this evaluation step would be computed using the mechanisms used to evaluate SPARQL queries.

Definition 7 (Applicable Authorizations). *Given a finite set of authorization* $\mathfrak{A} \in \mathcal{P}(\mathsf{Auth})$ *and a graph* $G \in \mathsf{BGP}$, *the function* ar *assigns to each triple* $t \in G$, *the subset of* applicable authorizations *from* $\mathfrak{A}$ *:*

$$\mathsf{ar}(G, \mathfrak{A})(t) = \{\mathfrak{a} \in \mathfrak{A} \mid \exists \theta \in [\![\mathsf{hb}(\mathfrak{a})]\!]_G . t = (\mathsf{head}(\mathfrak{a}))\theta\}$$

Example 5. Consider the graph $\mathsf{Cl}(G_0)$ shown in Fig. 1 and the set of authorizations $\mathfrak{A}$ shown in Table 1. The applicable authorizations on triple it_2 are computed as follows : $\mathsf{ar}(G, \mathfrak{A})(it_2) = \{\mathfrak{a}_5, \mathfrak{a}_6, \mathfrak{a}_9\}$. The mappings from $\mathsf{hb}(\mathfrak{a}_5)$, $\mathsf{hb}(\mathfrak{a}_6)$ and $\mathsf{hb}(\mathfrak{a}_9)$ to $\mathsf{Cl}(G_0)$ are illustrated by Fig. 2.

As exemplified above, there may exist some t such that the set $\mathsf{ar}(G, \mathfrak{A})(t)$ is not a singleton authorization. If the set of applicable authorizations is empty, then a solution to ensure that the decision function is total is to specify a *default decision*. When several authorizations with different effects are applicable, one has to specify a *conflict resolution strategy* that defines which of the effects has to be selected.

To prevent us from defining many extra parameters, arbitrarily fixing some conflict resolution strategies or running into considerations on conflict resolution

Table 1. Example of authorizations

$\mathfrak{a}_1$ = GRANT(?p ; :hasTumor ; ?t)
$\mathfrak{a}_2$ = DENY (?p ; rdf :type ; :cancerous)
$\mathfrak{a}_3$ = GRANT(?d ; :service ; ?s)
$\mathfrak{a}_4$ = GRANT(?d ; :treats ; ?p)
$\mathfrak{a}_5$ = DENY (?p ; :admitted ; ?s)
WHERE {(?s ; rdf :type ; :oncology)}
$\mathfrak{a}_6$ = GRANT(?p ; :admitted ; ?s)
$\mathfrak{a}_7$ = GRANT(?p ; rdfs :dom ; ?s)
$\mathfrak{a}_8$ = DENY (?s ; ?p ; :cancerous)
$\mathfrak{a}_9$ = DENY (?s ; ?p ; ?o)

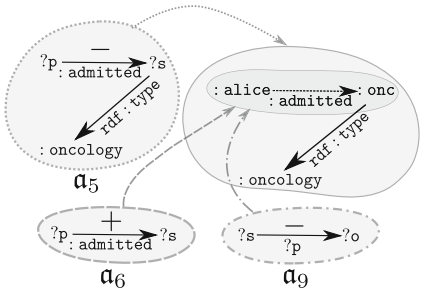


Fig. 2. Authorizations applicable to it_2

that are out of the scope of this paper, we abstract from the details of the concrete resolution strategies by assuming that there exists a choice function that, given a finite set of possibly conflicting authorizations, picks a unique one out. This design choice as well as the issues related to the modeling of classical conflict resolution strategies are discussed in Sect. 4.

Definition 8 (Conflict Resolution Function, Policy). *A* conflict resolution function ch *for authorizations is a function* $\mathsf{ch} \in \mathcal{P}(\mathsf{Auth}) \to \mathsf{Auth}$. *An* (authorization) policy P *is a pair* $P = (\mathfrak{A}, \mathsf{ch})$, *where* $\mathfrak{A}$ *is a finite set of authorizations and* ch *is a conflict resolution function, which satisfies the following coherence conditions:*

- *Closedness:* $\forall \mathfrak{A}' \subseteq \mathfrak{A}. \mathfrak{A}' \neq \emptyset \Rightarrow \mathsf{ch}(\mathfrak{A}') \in \mathfrak{A}'$
- *Totality:* $\forall G \in \mathsf{BGP}. \forall t \in G. \mathsf{ar}(G, \mathfrak{A})(t) \neq \emptyset$
- *Monotony:* $\forall \mathfrak{A} \subseteq \mathsf{Auth}. \mathsf{ch}(\mathfrak{A}) = \mathfrak{a} \Rightarrow (\forall \mathfrak{A}' \subseteq \mathfrak{A}. \mathfrak{a} \in \mathfrak{A}' \Rightarrow \mathsf{ch}(\mathfrak{A}') = \mathfrak{a})$

The subset of $\mathcal{P}(\mathsf{Auth}) \times (\mathcal{P}(\mathsf{Auth}) \to \mathsf{Auth})$ *that satisfies the above coherence conditions is denoted by* Pol.

The coherence conditions are properties which ensure that the conflict resolution functions behave well when applied to set of authorizations. The *Closedness* property guarantees that the selected rule is taken from the input. The *Totality* property avoids a corner case. We explain in Sect. 4 how to enforce *default decisions* that ensure this property. The *Monotony* property is more technical but it captures an intuitive requirement that is: *the conflict resolution function makes consistent choices*, which means its answer is kept the same when lesser choices are available.

Example 6. An example policy is $P = (\mathfrak{A}, \mathsf{ch})$ where $\mathfrak{A}$ is the set authorizations in Table 1 and ch is defined as follows. For all non-empty subset $\mathfrak{B}$ of $\mathfrak{A}$, $\mathsf{ch}(\mathfrak{B})$ is the first authorization (using syntactical order of Table 1) of $\mathfrak{A}$ that appears in $\mathfrak{B}$. For $\mathfrak{B} = \emptyset$, $\mathsf{ch}(\emptyset) = \mathfrak{a}_9$. *Closedness* and *Monotony* directly stem from the definition of ch. *Totality* stems from $\mathfrak{a}_9$, as it is applicable to any triple.

We are ready to give semantics to policies by composing the functions ar, ch and then effect in order to compute the authorized subgraph of a given graph.

Definition 9 (Policy Evaluation, Positive Subgraph). *Given a policy* $P = (\mathfrak{A}, \mathsf{ch}) \in \mathsf{Pol}$ *and a graph* $G \in \mathsf{BGP}$*, the set of authorized triples that constitutes the* positive subgraph *of* G *according to* P *is defined as follows, writing* G^+ *when* P *is clear from the context:*

$$G_P^+ = \{t \in G \mid (\mathsf{effect} \circ \mathsf{ch})(\mathsf{ar}(G, \mathfrak{A})(t)) = +\}$$

Example 7. Let us consider the policy $P = (\mathfrak{A}, \mathsf{ch})$ defined in Example 6, the graph G_0 of Fig. 1, and the triple $it_2 = (\texttt{:alice}; \texttt{:admitted}; \texttt{:onc})$. As we can see in Fig. 2, $\mathsf{ar}(\mathsf{Cl}(G_0), \mathfrak{A})(it_2) = \{\mathfrak{a}_5, \mathfrak{a}_6, \mathfrak{a}_9\}$. Since $\mathfrak{a}_5$ is the first among authorization in Table 1 and its effect is -, we deduce that $it_2 \notin \mathsf{Cl}(G_0)_P^+$. By applying a similar reasoning on all triples in $\mathsf{Cl}(G)$, we obtain $\mathsf{Cl}(G_0)_P^+ = \{et_1, et_3, et_4, et_5\}$. Note that et_2 is not authorized.

3.2 Consistency Property

The inference rules which are applied to a graph reflect the particular knowledge conveyed by the graph. Hence, the real semantics of a graph are represented by its closure, regardless it is materialized or not. Thus, information leakage has to be considered in the closure of a graph, rather than considering only the base graph which is under control of a trusted RDF store. A malicious user who knows the inference rules could use a local reasoner and apply the inference rules over his accessible triples to infer triples she/he is not supposed to access. To illustrate this issue, consider the following example.

Example 8. Assume a of inference rules $\mathbf{R} = \{\mathbf{RDom}, \mathbf{RAdm}\}$, as shown in Example 3. We want to apply the policy defined in Example 7 on the graph $\mathsf{Cl}_{\mathbf{R}}(G_0)$ of Fig. 1. According to Example 7, the authorized subgraph is $(\mathsf{Cl}_{\mathbf{R}}(G_0))_P^+ = \{et_1, et_3, et_4, et_5\}$. If one computes the closure of $(\mathsf{Cl}_{\mathbf{R}}(G_0))_P^+$, she/he obtains $(\mathsf{Cl}_{\mathbf{R}}(G_0))_P^+ \cup \{it_1, it_2\}$. Whereas the policy states that triples it_1 and it_2 must be denied, they are deduced from the authorized subgraph, hence the information leakage.

We formally characterize the issue that arises when inference rules produce facts that would have been forbidden otherwise. This issue occurs when the positive subset of a closed graph is not, itself, closed.

Definition 10 (Consistency between Rules and Policies). *An authorization policy* $P = (\mathfrak{A}, \mathsf{ch})$ *is* consistent *w.r.t. a set of inference rules* $\mathbf{R}$ *if, for any graph* $G \in \mathsf{BGP}$*, the following holds:*

$$\mathsf{Cl}_{\mathbf{R}}((\mathsf{Cl}_{\mathbf{R}}(G))_P^+) = (\mathsf{Cl}_{\mathbf{R}}(G))_P^+$$

The consistency property has to hold for all graphs. Therefore, it does not have to be checked when stored graphs are updated, but solely at the policy design-time or when the inference rules change. Given that the stored graphs are updated on a regular basis, we consider that policies and inference rules are more stable over time.

4 Building Policies

First, we illustrate the applicability of policies as defined in Definition 8 by showing how to construct motivating conflict resolution functions. Then, we show that the default decisions and common conflict resolution strategies can be modeled in our framework. In particular, we illustrate how the *Most Specific Takes Precedence* (MSTP) principle can be defined.

First of all, we notice that if there exists a *total order* denoted by $\preccurlyeq$ between authorizations of a set $\mathfrak{A}$, we can construct its associated conflict resolution function $\mathsf{min}_{\preccurlyeq}$ that selects the smallest element from a subset $\mathfrak{B} \subseteq \mathfrak{A}$. The *Closedness* and the *Monotony* conditions of Definition 8 are satisfied by construction. There are several ways to equip $\mathfrak{A}$ with a total order. For instance, the administrator can explicitly assign a unique prevalence level to each authorization or she/he can rely on the *syntactical order*. When one writes a set of authorizations such as the one shown in Table 1, there is a total order given by the order of the statements. The syntactical order is always available and it is used, for example, in firewalls, so that no ambiguity arises.

4.1 Default Policy

A default policy is a decision that is selected when no other authorization is applicable that is $\mathsf{ar}(G, \mathfrak{A})(t) = \emptyset$. Such a default policy can either be *deny by default* or *permit by default*. In order to respect the *Totality* coherence condition of Definition 8, we cannot simply apply a default *decision*. However, we have to identify a default *authorization*. The following lemma shows that the *Totality* condition can be ensured by adding a *universal authorization* which is applicable to any triple.

Lemma 1. *Let* $\mathfrak{A}$ *be a set of authorizations, the condition* $\forall G.\forall t \in G.\, \mathsf{ar}(G, \mathfrak{A})(t) \neq \emptyset$ *is equivalent to* $\exists \mathfrak{a}_{\mathrm{u}} \in \mathfrak{A}.\forall G.\forall t \in G.\mathfrak{a}_{\mathrm{u}} \in \mathsf{ar}(G, \mathfrak{A})(t)$.

We enforce the default policy by adding a universal authorization such as authorization $\mathfrak{a}_9$ as shown in Table 1. There may be several different universal authorizations in the set $\mathfrak{A}$. Therefore, conflicts will be systematically triggered. Even though it is formally possible to have several universal authorizations, we can assume that such a rule is *unique*. Note that the addition of a default rule at the end of a rule set is standard practice in firewall policies.

4.2 Precedence Strategies

The *Denials Take Precedence* (DTP) principle resolves conflicts by stating that the negative authorizations prevail over the positive ones; the *Permissions Take Precedence* (PTP) principle being its dual. The idea to capture the DTP (resp. PTP) strategy is to transform a policy $P = (\mathfrak{A}, \mathsf{ch})$ into a policy $P^{\text{-}} = (\mathfrak{A}, \mathsf{ch}^{\text{-}})$ where $\mathsf{ch}^{\text{-}}$ privileges negative (resp. positive) effects. Considering the previous discussion on default policies, we assume that there is a unique universal authorization $\mathfrak{a}_{\mathrm{u}} \in \mathfrak{A}$. As $\mathfrak{a}_{\mathrm{u}}$ is assumed to be a default authorization, we require that

$\mathfrak{B} \setminus \{\mathfrak{a}_u\} = \emptyset$ if and only if $\mathsf{ch}(\mathfrak{B}) = \mathfrak{a}_u$. Remind that $\mathfrak{B}^-$ (resp. $\mathfrak{B}^+$) is the subset of $\mathfrak{B}$ with a negative (resp. positive) effect. With $\mathfrak{B} \subseteq \mathfrak{A}$, the ch^- function is formally defined as follows:

$$\mathsf{ch}^-(\mathfrak{B}) = \begin{cases} \mathsf{ch}(\mathfrak{B}^- \setminus \{\mathfrak{a}_u\}) & \text{if } \mathfrak{B}^- \setminus \{\mathfrak{a}_u\} \neq \emptyset & (1) \\ \mathsf{ch}(\mathfrak{B}^+ \setminus \{\mathfrak{a}_u\}) & \text{if } \mathfrak{B}^- \setminus \{\mathfrak{a}_u\} = \emptyset \wedge \mathfrak{B}^+ \setminus \{\mathfrak{a}_u\} \neq \emptyset & (2) \\ \mathfrak{a}_u & \text{if } \mathfrak{B} \setminus \{\mathfrak{a}_u\} = \emptyset & (3) \end{cases}$$

Similarly, the dual function ch^+ is defined by flipping + and - in the definition of P^-. The next lemma ensures that the construction is correct.

Lemma 2 (Correctness of P^-). *Given $P = (\mathfrak{A}, \mathsf{ch})$ a policy according to Definition 8 with a unique universal authorization $\mathfrak{a}_u \in \mathfrak{A}$ such that $\forall \mathfrak{B} \subseteq \mathfrak{A}.\, \mathsf{ch}(\mathfrak{B}) = \mathfrak{a}_u \Rightarrow \mathfrak{B} \setminus \{\mathfrak{a}_u\} = \emptyset$, the structure $P^- = (\mathfrak{A}, \mathsf{ch}^-)$ is a policy as well.*

Example 9. Consider the graph $\mathsf{Cl}(G_0)$ shown in Fig. 1 and the set of authorizations $\mathfrak{A}$ shown in Table 1. Let us consider the authorizations applicable to triple et_1, that is $\mathsf{ar}(\mathsf{Cl}(G_0), \mathfrak{A})(et_1) = \{\mathfrak{a}_7, \mathfrak{a}_8, \mathfrak{a}_9\}$. If we consider the ch given in Example 6, that is, the syntactical order, authorization $\mathfrak{a}_7$, a positive one, is selected. However, with the DTP construction, we have that $\mathsf{ch}^-(\{\mathfrak{a}_7, \mathfrak{a}_8, \mathfrak{a}_9\}) = \mathsf{ch}(\{\mathfrak{a}_8\}) = \mathfrak{a}_8$.

4.3 Most Specific Takes Precedence (MSTP)

The MSTP strategy *partially* solves conflicts by choosing most specific authorizations first, then remaining conflicts are solved afterwards. This strategy is particularly adequate to capture exceptions in policies in a natural way. For instance, in Table 1, the authorization $\mathfrak{a}_5$ that denies admissions to oncology service is an exception to the authorization $\mathfrak{a}_6$ which allows admissions in general. According to the MSTP strategy, $\mathfrak{a}_5$ should prevail over $\mathfrak{a}_6$.

We say that an authorization $\mathfrak{a}_1$ *is more specific* than authorization $\mathfrak{a}_2$ denoted by $\mathfrak{a}_1 \sqsubseteq \mathfrak{a}_2$ when the underlying graph pattern of $\mathfrak{a}_2$ can be matched to the one of $\mathfrak{a}_1$ with the restriction that the head of $\mathfrak{a}_2$ is mapped to the head of $\mathfrak{a}_1$. More formally, $\mathfrak{a}_1 \sqsubseteq \mathfrak{a}_2 \equiv \exists \theta.\, \mathsf{hb}(\mathfrak{a}_2)\theta \subseteq \mathsf{hb}(\mathfrak{a}_1) \wedge \mathsf{head}(\mathfrak{a}_2)\theta = \mathsf{head}(\mathfrak{a}_1)$.

Clearly, the identity substitution makes the $\sqsubseteq$ relation reflexive and composition of substitution makes it transitive. Therefore, it is a preorder. We can define a function $\mathsf{mins}_{\sqsubseteq}$, from sets of authorizations to sets of authorizations, which keeps the most specific ones: $\mathsf{mins}_{\sqsubseteq}(\mathfrak{A}) = \{\mathfrak{a} \in \mathfrak{A} \mid \forall \mathfrak{a}' \in \mathfrak{A}.\ \mathfrak{a}' \sqsubseteq \mathfrak{a} \Rightarrow \mathfrak{a}' \sqsupseteq \mathfrak{a}\}$.

At this stage, the pair $(\mathfrak{A}, \mathsf{mins}_{\sqsubseteq})$ is not a policy yet: $\mathsf{mins}_{\sqsubseteq}$ is ambiguous. Therefore, it does not comply with coherence conditions of Definition 8. However, we can rely on the previous constructions for the DTP precedence strategy to define a more precise policy, by composing $\mathsf{mins}_{\sqsubseteq}$ with $\mathsf{min}^-_{\preccurlyeq_{lex}}$ (resp. $\mathsf{min}^+_{\preccurlyeq_{lex}}$ for PTP), where $\mathsf{min}_{\preccurlyeq_{lex}}$ is the conflict resolution function using the syntactical order. Finally, we obtain the structure $P = (\mathfrak{A}, \mathsf{min}^-_{\preccurlyeq_{lex}} \circ \mathsf{mins}_{\sqsubseteq})$ which is a fully-fledged policy.

Algorithm 1. Algorithm for enumerating inconsistency patterns

Input: a set of inference rules $\mathbf{R}$, an authorization policy $P = (\mathfrak{A}, \mathsf{ch})$
Output: a collection $BGPs$ of counterexample basic graph patterns

function RdfLeaks($\mathbf{R}$, P)
  $BGPs \leftarrow \emptyset$
  for all $\mathbf{r} = (tp \leftarrow tp_1, \ldots, tp_k) \in \mathbf{R}$ **do**
    for all $(\mathfrak{a}_1, \ldots, \mathfrak{a}_k, \mathfrak{a}) \in \mathfrak{A}^{+^k} \times \mathfrak{A}^-$ **do**
      let $\rho_1, \ldots, \rho_k, \rho$ be renaming substitutions for $\mathfrak{a}_1, \ldots, \mathfrak{a}_k, \mathfrak{a}$
      let $(ha_1, \ldots, ha_k, ha) = (\mathsf{head}(\mathfrak{a}_1)\rho_1, \ldots, \mathsf{head}(\mathfrak{a}_k)\rho_k, \mathsf{head}(\mathfrak{a})\rho)$
      if $\exists \mu = \mathsf{mgu}((ha_1, \ldots, ha_k, ha), (tp_1, \ldots, tp_k, tp))$ **then**
        let $B = \bigcup_{i=1}^{k} \mathsf{hb}(\mathfrak{a}_i)\rho_i\mu \cup \mathsf{hb}(\mathfrak{a})\rho\mu$
        if $\{(tp_1)\mu, \ldots, (tp_k)\mu\} \subseteq (\mathsf{Cl}_{\mathbf{R}}(B))^+_P$ **and** $(tp)\mu \notin (\mathsf{Cl}_{\mathbf{R}}(B))^+_P$ **then**
          $BGPs \leftarrow BGPs \cup \{B\}$
        end if
      end if
    end for
  end for
  return $BGPs$
end function

Example 10. Given a policy $P = (\mathfrak{A}, \mathsf{min}^{\text{-}}_{\preccurlyeq_{lex}} \circ \mathsf{mins}_{\sqsubseteq})$, the selected authorization for the triple it_2 is computed as follows : $(\mathsf{min}^{\text{-}}_{\preccurlyeq_{lex}} \circ \mathsf{mins}_{\sqsubseteq})(\mathsf{ar}(G, \mathfrak{A})(it_2)) = (\mathsf{min}^{\text{-}}_{\preccurlyeq_{lex}} \circ \mathsf{mins}_{\sqsubseteq})(\{\mathfrak{a}_5, \mathfrak{a}_6, \mathfrak{a}_9\}) = \mathsf{min}^{\text{-}}_{\preccurlyeq_{lex}}(\{\mathfrak{a}_5\}) = \mathfrak{a}_5$. If we consider et_1, we have $\mathsf{ar}(G, \mathfrak{A})(et_1) = \{\mathfrak{a}_7, \mathfrak{a}_8, \mathfrak{a}_9\}$ and $\mathsf{mins}_{\sqsubseteq}(\{\mathfrak{a}_7, \mathfrak{a}_8, \mathfrak{a}_9\}) = \{\mathfrak{a}_7, \mathfrak{a}_8\}$: the most specific authorization is not unique. Therefore, we rely on $\mathsf{min}^{\text{-}}_{\preccurlyeq_{lex}}$ to finally select $\mathfrak{a}_8$.

5 Static Verification

In this section, we show a key property of the framework introduced so far: it is possible to check, without any knowledge of a base graph, if a policy is consistent w.r.t. a set of inference rules. In other words, we define Algorithm 1 that, given an authorization policy $P = (\mathfrak{A}, \mathsf{ch})$ and a set of inference rules $\mathbf{R}$, checks whether Definition 10 holds. In fact, Algorithm 1 is an enumeration algorithm and not a mere decision algorithm: it is constructive and finds all possible counterexamples to the consistency property.

The principle of Algorithm 1 is to find an inference rule $(tp \leftarrow tp_1, \ldots, tp_k) \in \mathbf{R}$ and related sets of authorizations $(\mathfrak{a}_1, \ldots, \mathfrak{a}_k, \mathfrak{a})$ such that $\mathfrak{a}$ is negative and its head is unifiable with tp and all authorizations $\mathfrak{a}_i$ for $i \in \{1, \ldots, k\}$ are positive and their heads are unifiable with $\{tp_1, \ldots, tp_k\}$. Pictorially:

$$\mathbf{r} = \frac{\underbrace{\mathsf{hb}(\mathfrak{a}_1)}_{tp_1} \ \ldots \ \underbrace{\mathsf{hb}(\mathfrak{a}_k)}_{tp_k}}{\overbrace{\mathsf{hb}(\mathfrak{a})}^{tp}} \text{ with } \mathfrak{a}_i \in \mathfrak{A}^+ \text{ and } \mathfrak{a} \in \mathfrak{A}^-$$

Let us consider the graph B built by considering the union of the underlying graphs $\mathsf{hb}(\mathfrak{a}_1) \dots \mathsf{hb}(\mathfrak{a}_k)$ and $\mathsf{hb}(\mathfrak{a})$, properly renamed and unified. By construction, the inference rule $\mathbf{r}$ is applicable, thus $B \subsetneq \mathsf{Cl}_{\mathbf{R}}(B)$. Moreover, all authorizations are applicable as well. On the one hand, triples tp_1 to tp_k are authorized by some positive authorizations. On the other hand, tp is inferred using rule $\mathbf{r}$ but is forbidden by authorization $\mathfrak{a}$: an inconsistency.

The key idea that ensures the completeness of Algorithm 1 is that all counterexamples of the consistency property have to arise this way. Theorems 1 and 2 formally state the correctness of the algorithm: P is not consistent w.r.t. $\mathbf{R}$ *if and only if* Algorithm 1 returns a non empty collection. We rely on the usual definitions of unifiers and most general unifiers (mgu) as stated by Martelli and Montanari for instance, [9].

Theorem 1 (Soundness of Algorithm 1). *If Algorithm 1 returns a non empty collection then P is not consistent w.r.t. $\mathbf{R}$.*

Theorem 2 (Completeness of Algorithm 1). *Given a basic graph pattern G, if $\mathsf{Cl}_{\mathbf{R}}((\mathsf{Cl}_{\mathbf{R}}(G))^+_P) \neq (\mathsf{Cl}_{\mathbf{R}}(G))^+_P$, then there exists a basic graph pattern $B \in$ RdfLeaks$(\mathbf{R}, P)$ such that $[\![B]\!]_{\mathsf{Cl}_{\mathbf{R}}(G)} \neq \emptyset$.*

Theorem 1 holds by construction: Line 9 of Algorithm 1 ensures that B is a counterexample. Next, we prove Theorem 2 and discuss counterexample usage.

5.1 Main Theorem

To show that Theorem 2 holds, we first introduce two lemmas. Intuitively, Lemma 3 ensures that the Definition 7 of applicable authorization behaves well according to instantiation of graphs. Lemma 4 is its counterpart for the closure of a graph according to a set of inference rules.

Lemma 3. *Let $P = (\mathfrak{A}, \mathsf{ch})$ be an authorization policy, $B, G \in \mathsf{BGP}$ are basic graph patterns, and η is a substitution such that $B\eta \subseteq G$. For any $t \in B$, $\mathsf{ar}(B, \mathfrak{A})(t) \subseteq \mathsf{ar}(G, \mathfrak{A})((t)\eta)$.*

Lemma 4. *Let $P = (\mathfrak{A}, \mathsf{ch})$ be an authorization policy, $\mathbf{R}$ is a set of inference rules, $B, G \in \mathsf{BGP}$ are basic graph patterns, and η is a substitution such that $B\eta \subseteq G$. For any $t \in \mathsf{Cl}_{\mathbf{R}}(B)$, $(t)\eta \in \mathsf{Cl}_{\mathbf{R}}(G)$.*

Proof (Sketch of Proof of Theorem 2). Let G^{ex} be a counterexample graph as in the hypothesis of Theorem 2. First, we note that if a graph is not closed $\mathsf{Cl}_{\mathbf{R}}(G) \neq G$ then there are some triples not in G that are produced at the first step of the closure algorithm. By applying it to $(\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^+_P$, we know that there exists a triple $t^{ex} = (tp)\sigma$ produced by some rule $\mathbf{r} = (tp \leftarrow tp_1, \dots, tp_k) \in \mathbf{R}$ with $(tp_i)\sigma \in (\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^+_P$. By making hypothesis on $(\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^+_P$ and t^{ex}, we build the tuple $(\mathfrak{a}_1, \dots, \mathfrak{a}_k, \mathfrak{a})$ of authorizations that were selected by ch for tp_1, $\dots$, tp_k and tp. Then, by considering the heads of these authorizations, we can construct a unifier μ' between $\mathbf{r}$ and the authorizations once the authorizations

Table 2. Corrected authorization policy

$\mathfrak{a}_1$ = GRANT(?p ; :hasTumor ; ?t)	$\mathfrak{a}_5$ = DENY (?p ; :admitted ; ?s) WHERE {(?s ; rdf :type ; :oncology)}
$\mathfrak{a}_2$ = DENY (?p ; rdf :type ; :cancerous)	$\mathfrak{a}_6$ = GRANT(?p ; :admitted ; ?s)
$\mathfrak{a}_3$ = GRANT(?d ; :service ; ?s)	$\mathfrak{a}_8$ = DENY (?s ; ?p ; :cancerous)
$\mathfrak{a}'_3$ = DENY (?d ; :treats ; ?p) WHERE {(?d ; :service ; ?s), (?s ; rdf :type ; :oncology)}	$\mathfrak{a}_7$ = GRANT(?p ; rdfs :dom ; ?s)
$\mathfrak{a}_4$ = GRANT(?d ; :treats ; ?p)	$\mathfrak{a}'_8$ = GRANT (?x0 ; rdf :type ; ?x1)
	$\mathfrak{a}_9$ = DENY (?s ; ?p ; ?o)

are renamed. If there exists a unifier, so does the most general one, say μ, thus the condition at Line 7 is satisfied.

We construct B at Line 8 and consider its evaluation on $\mathsf{Cl}_{\mathbf{R}}(G^{ex})$. We know that $\mathsf{Cl}_{\mathbf{R}}(G^{ex})$ contains an instance of B because of μ and μ'. Using Lemmas 3 and 4 and the *Monotony* condition of Definition 8, we conclude that authorizations $(\mathfrak{a}_1, \ldots, \mathfrak{a}_k, \mathfrak{a})$ are also the ones selected by ch for the triples $(tp_1)\mu, \ldots (tp_k)\mu$, and $(tp)\mu$ in $\mathsf{Cl}_R(B)$. This means that the condition in Line 9 evaluates to true and B is in the result.

5.2 Understanding the Counterexamples

As Algorithm 1 enumerates inconsistency patterns, its output can be used to correct access control policy. A proof of concept of the algorithm has been implemented in Prolog[2]. The methodology to correct an inconsistent policy is to iteratively apply the following two steps: (1) use Algorithm 1 to obtain counterexample graph patterns; (2) change the authorization policy to correct inconsistencies illustrated by these graph patterns. The iteration stops when the authorization policy is consistent w.r.t. the set of inference rules. We illustrate this methodology on the inference rules of Example 3 and the policy defined in Table 1 with syntactical order. After three iterations, no inconsistency subsists anymore. The complete policy once corrected is given in Table 2.

The first two runs point out interactions between rule **RDom** and predicate rdf :type. The policy can be corrected by adding authorization $\mathfrak{a}'_8$ and switching authorizations $\mathfrak{a}_7$ and $\mathfrak{a}_8$. We give more details about the third run that produces a single counterexample graph B = {(?d ; :service ; ?s), (?d ; :treats ; ?p), (?p; :admitted; ?s), (?s; rdf :type; :oncology)} which involves the rule **RAdm** together with authorizations $\mathfrak{a}_3$, $\mathfrak{a}_4$ and $\mathfrak{a}_5$. A first and simple solution would be to change the effect of authorization $\mathfrak{a}_4$ to deny access to triples matching (?d ; :treats ; ?p). However, such an authorization would be extreme while the counterexample suggests to add a finer authorization $\mathfrak{a}'_3$ just before $\mathfrak{a}_4$. Note that we can alternatively switch (?d ; :treats ; ?p) and (?d ; :service ; ?s) in $\mathfrak{a}'_3$, but such a choice should be discussed with the experts first. After adding $\mathfrak{a}'_3$, a final execution of the algorithm confirms that the new policy is consistent w.r.t {**RDom**, **RAdm**} as it returns no counterexample.

[2] http://liris.cnrs.fr/~tsayah/DBSEC2015/.

Another way of using the counterexamples is to keep the policy unchanged, but to check if they occur in the actual closed graph managed by the RDF store. By Theorem 2, if there is no such instance, no information leakage will occur. Thus, one could use each B produced by Algorithm 1 as an integrity constraint in the RDF store, thereby reject updates that may lead to information leakage.

6 Related Work

The importance of confidentiality problems have been recognized for long. As such, access control models for different data models data have been proposed. RDF graphs can be written in a standard XML format, but there can be many different syntactical expressions that denote the same graph. Thus access control models for XML are quite difficult to transpose, if feasible, when applied to RDF graphs [7]. The Datalog model extends the relational one with deductive rules, thus one may devise a transformation that encodes graphs and rules into a Datalog program that uses a unique 3-ary relation symbol for triples [12], and then rely on access control mechanisms for deductive databases, such as the one by Barker [3]. Unfortunately, it seems that problems that arise when dealing with RDF data, the information leakage in particular, has not received much attention from the database community. We argue this because RDF is thought to be openly used between independent web sources, with shared or even standardized inference rules. In contrast, the Datalog model is more centralized, with rules and data under the control of a single authority.

Several access control models related to RDF data *without* inference rules have been proposed [1,5,13]. Abel et al. [1] propose a query rewriting mechanism to enforce authorizations. The authors do not present the formal semantics of the authorization language and their conflict resolution strategies are hard-coded. Flouris et al. [5] propose an annotation based access control language with its formal semantics for fine-grained authorizations on RDF data. The definition of authorizations in this paper is clearly inspired by Flouris et al. However, they used a fixed set of conflict resolution strategies (deny/permit by default and deny/permit takes precedence) without *most specific takes precedence*. In constrast, we advocate a more liberal approach. These models are sources of inspiration, but the problems related to inference rules are not addressed.

Other approaches consider inference rules and use propagation techniques to compute authorizations that are applicable to inferred triples [8,10,15]. Reddivari et al. [15] propose an access control language for RDF stores that considers update operations. They use meta-rules to define conflict resolution strategies and default policies but they do not provide formal semantics of their language. A similar approach inspired from provenance which has been proposed by Lopes et al. [8], where each triple is annotated with a label and labels are propagated through inference rule with a fixed conflict resolution strategy. Papakonstantinou et al. [10] propose a flexible model that defines the access label of a triple as an algebraic expression. They considered a fixed subset of RDFS rules only, but not user-defined rules. To sum up, the label-based techniques may use more expressive authorization languages or may consider updates, however they need some base graphs and they do not consider the information leakage.

Jain et al. [7] propose a label-based propagation technique for RDF data. They propose an algorithm that detects unauthorized inferences where higher security triples may be inferred from lower security triples. Nevertheless, a graph is needed to detect such violation, and their conflict resolution strategies and the default strategy are hard-coded. In contrast, we favor static analysis without knowledge of the graph and allow more flexible conflict resolution strategies. It would be interesting to check if their technique could be used to parallelize the computation of applicable authorizations with closure.

As a concluding remark, the inference problem we consider in this paper is a particular case of a more general one that is instantiated to the RDF data model [4]. Other orthogonal methods developed to deal with the general case, e.g., statistical ones or dynamic monitoring, may complement our statical verification technique.

7 Conclusion

In this paper, we introduced a fine-grained access control model for RDF stores with inference capabilities. We showed how concrete resolution strategies, notably *most specific takes precedence*, can be instances of our abstract framework. Whereas some models allow or deny *queries*, we gave semantics to authorizations by means of the *authorized subgraph* of a base graph, doing so we are independent of a given query language. We formalized an information leakage problem that arises when inferred triples are computed out of the RDF store by a (potentially) malicious user. We showed that, whenever the inference system can be expressed in a set of Datalog-like rules without negation, this property can be statically verified at the time of writing the authorization policy without the need of a base graph. Dealing with other inference systems such as OWL reasoning has to be further investigated.

The main issue related to the performance about our enforcement model stems from the definition of the applicable authorizations function (Definition 7). We propose the following technique using quad store technology, which adds a fourth attribute to triples. Given a policy $P = (\mathfrak{A}, \mathsf{ch})$, for each $\mathfrak{a} \in \mathfrak{A}$, compute $[\![\mathsf{hb}(\mathfrak{a})]\!]_G$. Then, we add authorization $\mathfrak{a}$ to the fourth attribute of each triple $(\mathsf{head}(\mathfrak{a}))\theta$ produced by some θ in $[\![\mathsf{hb}(\mathfrak{a})]\!]_G$. This technique assumes that the fourth attribute can be used to store the sets of identifiers, by means of the named graphs. This implementation is an ongoing work.

As for future work, we will study alternatives to the existence of a total order between authorizations to build the ch function. We plan to relax this condition with a user-defined partial order on authorizations. In order to build the ch function, an interesting perspective is to define it using the meet operator of a lower semilattice that extends the given partial order.

Additionally, we will compare our policy model against the existing ones. We envision to translate some well-known policy languages, e.g., XACML, into our formalism. As other models' semantics are usually expressed in terms of allowed or denied *queries* and not in terms of authorized *subgraphs*, verifying the correctness of such a translation would lead us to relate these different semantics.

As an example, for a query Q and an XACML policy X, the condition may be that $Q(G^{+}_{\alpha(X)}) = Q(G)$ if and only if $[\![X]\!]_X(Q) = \top$ where α is the translation function and $[\![\_]\!]_X$ is the interpretation function of XACML [14].

Finally, we plan to study the impact of RDF data updates, indeed, new issues arise from updates. For instance, a user may be allowed to insert a triple, but she/he may not be allowed to insert some of its consequences that can be inferred. We would like to characterize this problem with a new consistency property for updates, inspired by the one given in Sect. 3.2.

Acknowledgements. This work is supported by Thomson Reuters in the framework of the Partner University Fund project : "*Cybersecurity Collaboratory: Cyberspace Threat Identification, Analysis and Proactive Response*". The Partner University Fund is a program of the French Embassy in the United States and the FACE Foundation and is supported by American donors and the French government.

References

1. Abel, F., De Coi, J.L., Henze, N., Koesling, A.W., Krause, D., Olmedilla, D.: Enabling advanced and context-dependent access control in RDF stores. In: Aberer, K., et al. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 1–14. Springer, Heidelberg (2007)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995). http://webdam.inria.fr/Alice/
3. Barker, S.: Protecting deductive databases from unauthorized retrieval and update requests. Data Knowl. Eng. **43**(3), 293–315 (2002)
4. Farkas, C., Jajodia, S.: The inference problem: a survey. SIGKDD Explor. Newsl. **4**(2), 6–11 (2002)
5. Flouris, G., Fundulaki, I., Michou, M., Antoniou, G.: Controlling access to RDF graphs. In: Berre, A.J., Gómez-Pérez, A., Tutschku, K., Fensel, D. (eds.) FIS 2010. LNCS, vol. 6369, pp. 107–117. Springer, Heidelberg (2010)
6. Hayes, P., McBride, B.: RDF semantics. Technical report, W3C (2004)
7. Jain, A., Farkas, C.: Secure resource description framework: an access control model. In: SACMAT, pp. 121–129. ACM (2006)
8. Lopes, N., Kirrane, S., Zimmermann, A., Polleres, A., Mileo, A.: A logic programming approach for access control over RDF. In: ICLP, pp. 381–392 (2012)
9. Martelli, A., Montanari, U.: An efficient unification algorithm. ACM Trans. Program. Lang. Syst. **4**, 258–282 (1982)
10. Papakonstantinou, V., Michou, M., Fundulaki, I., Flouris, G., Antoniou, G.: Access control for RDF graphs using abstract models. In: SACMAT, pp. 103–112 (2012)
11. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009)
12. Polleres, A.: From SPARQL to rules (and back). In: WWW, pp. 787–796 (2007)
13. Rachapalli, J., Khadilkar, V., Kantarcioglu, M., Thuraisingham, B.: Towards fine grained RDF access control. In: SACMAT, pp. 165–176. ACM (2014)
14. Kencana Ramli, C.D.P., Nielson, H.R., Nielson, F.: The logic of XACML. In: Arbab, F., Ölveczky, P.C. (eds.) FACS 2011. LNCS, vol. 7253, pp. 205–222. Springer, Heidelberg (2012)
15. Reddivari, P., Finin, T., Joshi, A.: Policy-based access control for an RDF store. In: Policy Management for the Web workshop, WWW. pp. 78–81 (2005)