Selective disclosure and inference leakage problem in the Linked Data

Tarek Sayah

N° d'ordre NNT : 2016LYSE1156

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

École Doctorale ED512
Informatique et Mathématiques

Spécialité de doctorat : Informatique

Soutenue publiquement le 08/09/2016, par :
Tarek Sayah

Selective disclosure and inference leakage problem in the Linked Data

Devant le jury composé de :

Mé Ludovic, Professeur, Supélec - Cesson Sévigné	Président
Goasdoué François, Professeur des Universités, Université Rennes 1	Rapporteur
Nguyen Benjamin, Professeur des Universités, INSA Centre Val de Loire	Rapporteur
Roca Vincent, Chargé de recherche - HDR, Inria - Grenoble	Examinateur
Simon Eric, Architecte, SAP Business Objects - Levalois-Perret	Examinateur
Hacid Mohand-Saïd, Professeur des Universités, Université Lyon 1	Directeur de thèse
Coquery Emmanuel, Maître de conférences, Université Lyon 1	Co-directeur de thèse
Thion Romuald, Maître de conférences, Université Lyon 1	Co-directeur de thèse

UNIVERSITE CLAUDE BERNARD - LYON 1

Président de l'Université	**M. François-Noël GILLY**
Vice-président du Conseil d'Administration	M. le Professeur Hamda BEN HADID
Vice-président du Conseil des Etudes et de la Vie Universitaire	M. le Professeur Philippe LALLE
Vice-président du Conseil Scientifique	M. le Professeur Germain GILLET
Directeur Général des Services	M. Alain HELLEU

COMPOSANTES SANTE

Faculté de Médecine Lyon Est – Claude Bernard	Directeur : M. le Professeur J. ETIENNE
Faculté de Médecine et de Maïeutique Lyon Sud – Charles Mérieux	Directeur : Mme la Professeure C. BURILLON
Faculté d'Odontologie	Directeur : M. le Professeur D. BOURGEOIS
Institut des Sciences Pharmaceutiques et Biologiques	Directeur : Mme la Professeure C. VINCIGUERRA
Institut des Sciences et Techniques de la Réadaptation	Directeur : M. le Professeur Y. MATILLON
Département de formation et Centre de Recherche en Biologie Humaine	Directeur : Mme. la Professeure A-M. SCHOTT

COMPOSANTES ET DEPARTEMENTS DE SCIENCES ET TECHNOLOGIE

Faculté des Sciences et Technologies	Directeur : M. F. DE MARCHI
Département Biologie	Directeur : M. le Professeur F. FLEURY
Département Chimie Biochimie	Directeur : Mme Caroline FELIX
Département GEP	Directeur : M. Hassan HAMMOURI
Département Informatique	Directeur : M. le Professeur S. AKKOUCHE
Département Mathématiques	Directeur : M. le Professeur Georges TOMANOV
Département Mécanique	Directeur : M. le Professeur H. BEN HADID
Département Physique	Directeur : M. Jean-Claude PLENET
UFR Sciences et Techniques des Activités Physiques et Sportives	Directeur : M. Y.VANPOULLE
Observatoire des Sciences de l'Univers de Lyon	Directeur : M. B. GUIDERDONI
Polytech Lyon	Directeur : M. P. FOURNIER
Ecole Supérieure de Chimie Physique Electronique	Directeur : M. G. PIGNAULT
Institut Universitaire de Technologie de Lyon 1	Directeur : M. le Professeur C. VITON
Ecole Supérieure du Professorat et de l'Education	Directeur : M. le Professeur A. MOUGNIOTTE
Institut de Science Financière et d'Assurances	Directeur : M. N. LEBOISNE

A mes chers parents
A ma chère épouse et mes adorables filles
A mes frères et sœurs
A la mémoire de ma grand-mère paternelle
A mes amis et à la mémoire de mon ami *Fares Khentout*

Remerciements

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma reconnaissance.

Avant tout, j'adresse mes sincères remerciements à mon directeur de thèse *Mohand-Saïd Hacid* et à mes co-encadrants *Romuald Thion* et *Emmanuel Coquery*, pour leur soutien tout au long de ces trois années de thèse et pour leurs nombreux conseils qui m'ont été très précieux.

Je remercie vivement *François Goasdoué* et *Benjamin Nguyen* d'avoir rapporté mes travaux de thèse.

Je tiens à remercier également *Ludovic Mé*, *Vincent Roca* et *Eric Simon* d'avoir participé à mon jury et d'avoir accepté d'examiner mes travaux de thèse.

Je remercie également *Mohamed Frendi*, qui m'a encouragé à faire ce doctorat.

Je voudrais aussi exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur support moral et intellectuel tout au long de ma démarche, en particulier *Amine*, *Samir*, *Mehdi*, *Brahim* et *Fairouz*.

Merci à mon frère *Samir* pour son aide et ses encouragements.
Un grand merci à mes amis : *Kamel*, *Ahmed*, *Nadhir*, *Sayd*, *Lazhar*, *Ali* et *Fares*.
Merci à mon voisin et ami *Fouad*.

Je tiens à remercier également les membres du LIRIS pour leur accueil ainsi que pour leur ambiance chaleureuse tout au long de ma thèse en particulier *Brigitte* et *Isabelle*.

List of publications

[Sayah 2015] Tarek Sayah, Emmanuel Coquery, Romuald Thion and Mohand-Saïd Hacid. Inference Leakage Detection for Authorization Policies over RDF Data. In DBSec, USA 2015 (CORE Ranking : A).

[Sayah 2016] Tarek Sayah, Emmanuel Coquery, Romuald Thion and Mohand-Saïd Hacid. Access Control Enforcement for Selective Disclosure of Linked Data. In STM, Greece 2016 (Acceptance rate : 35%).

Résumé étendu en français

1 Contexte

Le Web est devenu l'outil principal de diffusion de l'information pour les organismes privés et publics, à travers lequel d'énormes quantités d'informations sont échangées chaque jour. Le web traditionnel est utilisé pour transmettre, recevoir et afficher des contenus encapsulés sous forme de documents. Ces documents étaient destinés à la consommation humaine et ne pouvait pas être compris par des machines. Le *Web sémantique* également appelé *Web de données* peut être conceptualisé comme une extension du Web actuel afin de permettre la création, le partage et la réutilisation intelligente des contenus web pour qu'ils soient lisibles par la machine.

Le *Linked Data* concerne tout simplement l'utilisation du Web pour créer des liens typés entre les données provenant de sources différentes. Techniquement, Linked Data réfère à un ensemble de bonnes pratiques pour l'édition et la connexion des données structurées sur le Web de sorte que ces données soient lisibles par la machine, leur sens est explicitement definie, elles sont liées à d'autres données externes, et être reliées à partir de données externes. Berners-Lee discute les quatre principes de base pour le Linked Data comme [BL06]:

1. Utiliser les URIs pour nommer les choses
2. Utiliser les URIs HTTP de telle sorte que les gens puissent chercher ces noms
3. Quand quelqu'un regarde une URI, fournir des informations utiles, en utilisant les normes (RDF, SPARQL)
4. Inclure des liens vers d'autres URIs, afin qu'ils puissent découvrir plus de choses

Un mouvement général "d'ouverture des données" (Linked Open Data) [1] est suivi par les institutions (ex., Enseignement Supérieur et Recherche en France [2]), les collectivités locales (ex., Villes de Lyon et de Grenoble [3]), les bases de données thématiques (ex., Dbtune, MusicBrainz [4]), les organizations (BBC dans ses systèmes internes de production de contenus [KSR+09]), géographiques et bien sûr généralistes (ex., GeoNames, Wikipedia [5])).

Lier des données distribuées à travers le Web nécessite un mécanisme standard pour spécifier l'existence et la signification des liens entre les éléments décrits dans ces données. Ce mécanisme est fourni par le format *Resource Description Framework* (RDF). La Figure 1 montre des ensembles de

[1] http://www.mckinsey.com/business-functions/business-technology/our-insights/open-data-unlocking-innovation-and-performance-with-liquid-information

[2] http://data.enseignementsup-recherche.gouv.fr

[3] http://data.grandlyon.com et http://data.beta.metropolegrenoble.fr

[4] http://dbtune.org and http://linkedbrainz.org

[5] http://www.geonames.org/ontology and http://wiki.dbpedia.org

données qui ont été publiés dans le format Linked Data[6]. La figure montre des datastores qui appartiennent à différents domaines de thématiques (gouvernement, publications, sciences de la vie, etc.). La taille du Web de données est estimé à plus de 3400 sources ouvertes pour un volume total de plus de 85 milliards de triplets(statements)[7], et plus de 8500 sources sur le portail de données ouvertes de l'Union Européenne[8].

RDF a gagné beaucoup d'attention ces dernières années, et par conséquent un nombre croissant d'ensembles de données sont maintenant représentés avec ce langage. Cette popularité a évidemment motivé certains travaux de recherche sur la conception de systèmes de bases de données adaptés et efficaces pour stocker efficacement, interroger et raisonner sur de grandes quantités de données RDF. Ces systèmes sont souvent appelés *Entrepôts RDF (RDF store)* ou *triplestore*. Plusieurs entrepôts RDF ont été développés pour traiter les données RDF, dont certains sont natifs alors que d'autres utilisent des backend SGBDR. Les entrepôt RDF natifs sont ceux qui sont mis en oeuvre à partir de zéro et qui exploitent le modèle RDF pour stocker efficacement et accéder aux données RDF, tels que 4Store[9], AllegroGraph[10], Stardog[11], Jena TDB [12] et Sesame[13]. Les entrepôts qui ne sont pas natif RDF tels que Virtuoso[14] et Jena SDB[15], utilisent une base de données relationnelle pour stocker les triplets. L'un des concepts clés de l'architecture du Web sémantique sont les *graphes nommés (named graphs)* qui est une simple extension du modèle de données RDF qui transforme les triplets RDF en *quads*. Les graphes nommés sont des ensembles de triplets identifiés par IRIs, permettant des descriptions sur ces ensembles comme des informations sur la provenance, le contexte et d'autres métadonnées.

Comme la demande pour les données et la gestion de l'information augmentent, il y a aussi un besoin crucial pour le maintien de la sécurité des sources de données, les applications, et systèmes d'information. Différents domaines d'études ont été introduits pour concevoir des approches appropriées capable de fournir des garanties de sécurité. Ces domaines d'études concernent l'authentification [Lam81], le contrôle d'accès [GW76], la cryptographie [DH76, RSA78] et l'audit [ABF$^+$04]. Suite à l'évolution de ces domaines, les systèmes de bases de données offrent de nouveaux paradigmes de sécurité pour faire face aux vulnérabilités induites par de nouvelles fonc-

[6] `http://lod-cloud.net/`

[7] `https://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets/LinkStatistics`

[8] `https://open-data.europa.eu/fr/data`

[9] `www.4store.com`

[10] `http://franz.com/agraph/allegrograph/`

[11] url http: //www.stardog. com /

[12] `https://jena.apache.org/documentation/tdb/index.html`

[13] `http://www.rdf4j.org/`

[14] `http://virtuoso.openlinksw.com/`

[15] `https://jena.apache.org/documentation/sdb/`

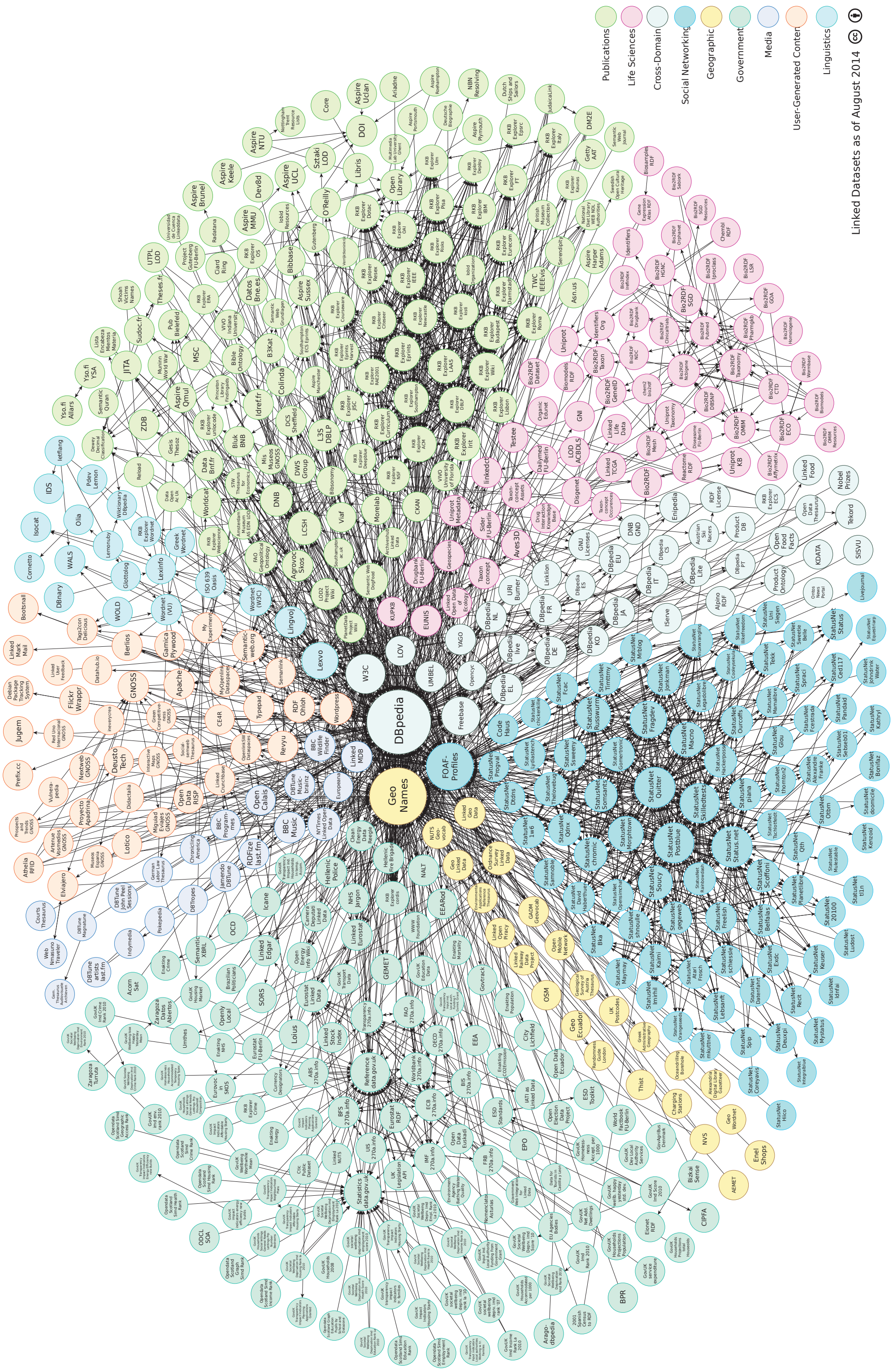


FIGURE 1 – Diagramme du Linked Data (30/08/2014)

tionnalités (par exemple, la distribution, l'hétérogénéité, l'autonomie et le raisonnement).

La protection des ressources contre les accès non autorisés est l'une des principales caractéristiques des systèmes d'aujourd'hui. L'un des principaux services de sécurité nécessaires pour assurer la protection des données est le contrôle d'accès. Ce dernier garantit qu'un utilisateur ne peut accéder qu'aux ressources auxquelles elle/il est autorisé.

Dans cette thèse, nous nous sommes concentré sur les problèmes de sécurité qui se posent principalement dans le contexte du Linked Data, en particulier l'exposition sélective des données RDF.

Notre objectif principal est d'encourager les entreprises et les organisations à publier leurs données RDF dans l'espace global de données liées. En effet, les données publiées peuvent être sensibles, et par conséquent, les fournisseurs de données peuvent être réticent à publier leurs informations, à moins qu'ils ne soient certains que les droits d'accès souhaités des différentes entités sont appliquées correctement à leurs données, et qu'aucune donnée sensible n'est révélée (par erreur). Donc la question de la sécurisation du contenu RDF et la garantie de l'exposition sélective de l'information à différentes catégories d'utilisateurs devient d'autant plus importante.

Paul Terry a écrit dans CTO Vision [16]: *To realize value from your data, you need to be able to share it among many stakeholders-internal lines of business, partners, researchers, and many others. But you also have a moral, ethical, and often legal, obligation to make sure that data is used responsibly. That means protecting individuals? privacy and assuring that their data is used only for legitimate purposes. As you gather more data from more parts of your organization, that gets very tricky very fast. [...] It quickly becomes clear that this is about more than simply checking a box for whether data is "secure". It?s incredibly valuable to have all that data in one place where it can be analyzed, but you need to assure that different types of stakeholders can see only the information they legitimately need, and no more.*

2 Énoncé du problème

Dans cette thèse, nous nous sommes intéressés à l'exposition sélective des données RDF et du problème de fuite d'inférence. Au cours des dernières années, le problème de contrôler l'accès aux données RDF a attiré une attention considérable à la fois la communauté de la sécurité et celle de base de données. Notre objectif était de définir une politique d'autorisations qui sera intégrée à l'entrepôt RDF pour contrôler l'exposition des données RDF. De plus, nous devons nous assurer que les données divulguées

[16] `https://ctovision.com/2015/12/big-data-unlocks-valuable-information-across-organizations-but-only-if-you-can-protect-it/`

ne peuvent pas être utilisées pour déduire des informations confidentielles. Dans ce contexte, nous étions intéressés par les problèmes suivants:

2.1 Exposition sélective des données RDF

Compte tenu de la nature sensible de l'information, les différentes parties de données RDF peuvent nécessiter des droits d'accès selon les privilèges du requérant. Dans cette thèse, nous nous sommes intéressés à l'exposition *sélective* des informations en fonction du contenu RDF, de qui le demande et sous quel contexte. La plupart des entrepôts RDF de nos jours offrent une protection au niveau graphe nommé. Par exemple, une telle fonctionnalité a été introduite dans 4store version 1.1.5 [17]. La sécurité au niveau graphe est disponible dans Virtuoso [18] ou aussi sur Stardog version 3.1 [19]. Le problème avec ces modèles est que les politiques de contrôle d'accès sont définies sur des graphes nommés qui doivent être créés par rapport aux politiques. Par exemple, si une politique stipule qu'une infirmière possède l'accès aux dossiers des patients, alors tous les triplets liés aux dossiers du patient doivent être réunis dans un seul graphe nommé sur lequel la politique est définie. En outre, les politiques complexes peuvent conduire à la création de plusieurs graphes nommés. Avoir un grand nombre de politiques complexes peut conduire l'administrateur à créer plusieurs graphes nommés qui pourraient être difficiles à gérer. De plus, un problème de redondance est soulevé puisque un triplet peut appartenir à plusieurs graphes nommés. Certains entrepôts RDF sont plus expressifs comme AllegroGraph [20] sur lequel l'accès est basé sur des motifs relatifs à un unique triplet [21]. Ceci permet la définition des autorisations simples telles que *interdire ou autoriser l'accès aux triplets représentant les dossiers des patients*. Cependant, des politiques plus expressives ne peuvent pas être spécifiés. Par exemple, une autorisation, telle que *Refuser l'accès aux dossiers des patients s'ils ont le cancer* ne peut pas être spécifiée.

Étant donné la nature ouverte du web dans lequel les données RDF sont publiées, le sujet pourrait ne pas être connu par le système avant l'envoi d'une requête. Par conséquent, l'exposition sélective ne peut pas compter que sur des politiques de contrôle d'accès traditionnelles basées sur l'identité ou le rôle. Le controle d'accès basé sur les attributs (Attribute Based Access Control (ABAC)) est un modèle qui permet plus de flexibilité en définissant des autorisations sur la base des attributs des sujets et objets. ABAC permet d'éviter la nécessité d'attribuer directement les autorisations à des sujets

[17] `http://4store.org/trac/wiki/GraphAccessControl`

[18] `http://docs.openlinksw.com/virtuoso/rdfgraphsecurity.html`

[19] `http://docs.stardog.com/#_security`

[20] `http://franz.com/agraph/allegrograph`

[21] `http://franz.com/agraph/support/documentation/v4/security.html#filters`

individuels avant leur demande d'effectuer une opération sur l'objet.

2.2 Problème de fuite d'inférence

Le deuxième enjeu est de veiller à ce que des informations sensibles ne puissent pas être déduites, en utilisant les règles d'inférence, une fois que les données ont été communiquées à l'utilisateur. Ce problème est connu dans la littérature de contrôle d'accès comme *problème d'inférence* [FJ02] appelé dans le manuscrit *problème de fuite d'inférence*. Selon le World Wide Web Consortium (W3C), l'inférence sur le Web sémantique en utilisant le Resource Description Framework (RDF) permet "*d'améliorer la qualité de l'intégration des données sur le Web en découvrant de nouvelles relations, et d'analyser automatiquement le contenu des données*". Les règles d'inférence sont utilisées pour obtenir de nouveaux triplets à partir de ceux qui sont explicitement affirmés dans un entrepôt RDF. En particulier, un ensemble de règles d'inférence connu par RDF *Schema* (RDFS) a été normalisé [HPS14]. Des modèles d'autorisation ont été proposés pour le contrôle des accès aux données RDF, à la fois en présence de règles d'inférence [RFJ05, LKZ$^+$12, PMF$^+$12, JF06] ou non [ADCH$^+$07, FFMA10, RKKT14]. Cependant, le problème est que cette capacité d'inférence peut être utilisée par un utilisateur (malveillant) pour déduire des données sensibles à partir de données publiques.

Pour illustrer le problème de fuite d'inférence, supposons que les triplets RDF indiquant que quelqu'un a un cancer sont étiquetés comme confidentiels (ex.triplets similaires à (?`p` ; `rdf :type` ; `:Cancerous`) avec ?`p` désignant une personne), tandis que ceux indiquant qu'une personne a une tumeur sont publics au sein d'un hôpital (ex. triplets de la forme (?`p`; `:hasTumor`; ?`t`)). S'il existe un triplet public indiquant que le domaine du prédicat `:hasTumor` est `:Cancerous` (ex. (`:hasTumor`; `rdfs :domain`; `:Cancerous`)), alors, en utilisant la règle RDFS qui fait correspondre le domaine d'un prédicat au type de ses sujets, des informations sensibles peuvent être déduites des triplets publics. La situation est encore pire lorsque le système de déduction est enrichi avec des règles définies par l'utilisateur.

2.3 Application et performance

Des enjeux particuliers découlent de l'application du contrôle d'accès. Le premier enjeu est l'impact de l'application du modèle de contrôle d'accès sur la performance du système: nous devons nous assurer que l'application du contrôle d'accès occasionne un faible surcoût sur la performance des entrepôts RDF. Le deuxième enjeu sont les mécanismes nécessaires pour appliquer le contrôle d'accès. Le mécanisme d'application doit être déployable dans l'entrepôt RDF avec des mécanismes supplémentaires minimes et, idéalement, sans altération de ses composants internes.

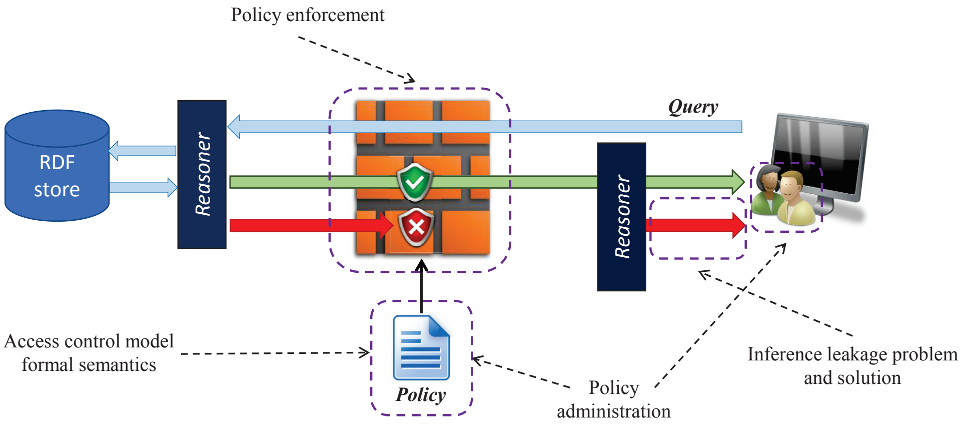

Policy enforcement
Query
RDF store
Reasoner
Reasoner
Access control model formal semantics
Policy
Policy administration
Inference leakage problem and solution

FIGURE 2 – Contributions principales

3 Étude des travaux connexes

Une première étude de l'état de l'art a conduit à la définition des critères d'évaluation des travaux connexes dans le contrôle d'accès aux données RDF, y compris l'expressivité de spécification des politiques, la résolution des conflits générés par des décisions contradictoires et la vérification des inférences non autorisées. Nous nous sommes appuyés sur ces critères pour analyser les travaux réalisés dans le domaine du contrôle d'accès aux données RDF. L'étude a permis de déterminer des critères bien traités tels que les actions supportées, l'expressivité des objets et la protection des triplets explicites et implicites. D'autres critères ont été peu considérés, y compris la résolution des conflits, l'expressivité des sujets et le problème de fuite inférence. Nous nous sommes intéressés sur ces derniers critères pour les contributions de cette thèse.

4 Contributions

Suite à une analyse détaillée des modèles existants de contrôle d'accès aux données RDF, nos contributions ont concerné quatre aspects, illustrés par la Figure 2.

4.1 Conception d'un nouveau modèle de contrôle d'accès

Nous avons conçu un nouveau modèle de contrôle d'accès pour les données RDF appelé `AC4RDF` pour *Access Control For the Resource Description Framework*. Nous avons défini la sémantique formelle d'un langage déclaratif basé sur les motifs de graphe élémentaires du langage SPARQL, afin de faciliter l'intégration du modèle dans les systèmes concrets. La sémantique du modèle est définie par le *sous-graphe positif* (autorisé) à partir du graphe de base. La requête de l'utilisateur est évaluée sur son sous-graphe positif ne contenant que les triplets accessibles. Notre modèle est ainsi complètement indépendant du langage de requêtes utilisé. Notre langage permet de définir des autorisations positives et négatives afin de traiter les exceptions des droits d'accès qui pourraient se produire dans la vie réelle. L'utilisation des autorisations négatives donne lieu au traitement des conflits qui se produisent lorsque le même objet est autorisé par une autorisation et refusé par une autre. Des stratégies de résolution des conflits sont utilisées pour résoudre de tels conflits en sélectionnant les autorisations préférées à appliquer par rapport à certaines propriétés des autorisations. Alors que la plupart des travaux codent en dur leurs stratégies, d'autres les définissent comme des paramètres qui sont fixés par l'administrateur lors de la conception de la politique. Nous proposons une approche plus libérale en définissant notre politique en utilisant une *fonction abstraite* de résolution de conflits qui est fournie par l'administrateur. Nous montrons comment coder les stratégies classiques telles que la *First Applicable*, *Denials Take Precedence* et la *Permissions Take Precedence* en plus de stratégies plus élaborées telle que la *Most Specific Takes Precedence*.

4.2 Problème de fuite d'inférence et solutions

Les modèles de contrôle d'accès classiques offrent une protection contre les accès *directs* à des informations sensibles. Cependant, les accès *indirects* à des informations sensibles peuvent encore être possible via des règles inférences. Dans le contexte du Linked Data, le problème se produit lorsque les triplets implicites sensibles peuvent être déduite à partir de triplets non sensibles à l'aide de règles d'inférence connues. Notre deuxième contribution a consisté à définir formellement la *propriété de cohérence* qui capture le problème des fuites d'inférence qui se pose lorsque des triplets confidentiels sont déduits à partir de triplets autorisés. Cette propriété garantit que les informations confidentielles ne peuvent pas être déduites à partir d'informations non confidentielles par rapport à un ensemble de règles d'inférence. Pour résoudre ce problème, nous proposons un algorithme de vérification statique, prouvé correct et complet, qui permet de détecter si une politique est cohérente quel que soit le graphe RDF utilisé. En cas d'incohérence de la politique, l'algorithme génère un ensemble de motifs de graphes contre-exemples qui représentent des modèles sur lesquels la politique présente un problème de fuite d'inférence. Ces graphes contre-exemples peuvent être utilisés par l'administrateur pour corriger la politique. Nous proposons une méthode itérative qui permet à l'administrateur de corriger la politique en utilisant l'algorithme. Nous montrons une autre alternative d'utilisation des graphes contre-exemples qui consiste à les utiliser comme contraintes d'intégrité lors des mises à jour du graphe de base.

4.3 Administration des politiques

Les politiques du modèle `AC4RDF` sont définies sans spécifier le sujet pour lequel les autorisations sont assignés. Ce qui permet d'utiliser n'importe quel modèle de contrôle d'accès en amont pour mapper les utilisateurs à leurs autorisations assignées. Nous proposons un langage de contrôle d'accès de haut niveau qui permet la définition de politiques globales qui sont ensuite compilées en politiques concrètes du modèle `AC4RDF`. Nous avons choisi de définir nos politiques sur la base des attributs d'utilisateur suivant l'approche Attribute Based Access Control (ABAC), où l'accès aux ressources protégées est basé sur un utilisateur ayant des attributs spécifiques (par exemple le nom, le rôle, la date de naissance, adresse, numéro de téléphone, etc). Nous avons défini la syntaxe et la sémantique d'un langage inspiré par XACML en définissant les principaux composants de la solution proposée et en montrant comment la politique de l'utilisateur est générée et appliquée. Intuitivement, une politique globale peut être représentée dans une structure d'arbre où les noeuds intermédiaires représentent les politiques, les feuilles représentent les autorisations et les arêtes sont étiquetées avec des cibles représentant des conditions à base d'attributs. Les conditions sont évaluées en utilisant des paires clé/valeur représentant les attributs fournis par les sujets. Celui-ci fournit ses attributs dans une demande qui est évaluée sur l'arbre de la politique globale pour déterminer les autorisations qui lui sont attribuées. Sur la base de ces autorisations, le sous-graphe positif du sujet est calculé, et la requête est évaluée sur ce sous-graphe ne retournant que ses triples accessibles.

4.4 Application du modèle de contrôle d'accès et expérimentations

Cette contribution consiste en la proposition d'un cadre d'application basé sur les annotations pour le modèle `AC4RDF`. L'idée est de matérialiser les autorisations applicables à chaque triplet dans un bitset qui est utilisé pour annoter ce triplet. De même, des bitsets sont aussi attribués aux sujets représentant le sous-ensemble des autorisations qui leur est assigné. Pour calculer le sous-graphe positif du sujet, la fonction de résolution de conflits est appliquée sur l'intersection entre l'ensemble des autorisations applicables au triplet et celles assignées au sujet. Ce qui revient à faire un *et* logique entre les bitsets du triplet et du sujet. Le système évalue ensuite la requête du sujet sur son sous-graphe positif. Pour stocker les bitsets, nous utilisons la position du nom de graphe. Faisant ainsi, nous n'avons pas besoin de mécanismes supplémentaires pour appliquer notre modèle vu que la majorité des entrepôts RDF actuels supportent les graphes nommés. Nous avons implémenté notre solution sur Jena TDB[22] et effectué des expérimentations pour

[22] `https://jena:apache:org/documentation/tdb`

calculer les surcoûts de l'intégration du modèle sur la performance du système. Nos expérimentations ont montré que notre implémentation entraîne un surcoût raisonnable durant l'exécution (environ + 50%) par rapport à la solution optimale qui consiste à matérialiser le sous-graphe positif. Il faut noter que cette dernière solution n?assure pas le passage à l?échelle puisqu?il faut matérialiser le sous-graphe positif de chaque utilisateur.

5 Organisation du manuscrit

Ce manuscrit comporte 3 chapitres précédés d'une introduction et suivi d'une conclusion. Il comporte également une annexe qui contient des définitions et lemmes techniques additionnels ainsi que la liste des requêtes utilisées dans les expérimentations.

Chapitre 1 : Introduction

L'introduction décrit le contexte et les différents défis auxquels nous nous sommes intéressés dans cette thése et les solutions qu'on a proposées. Dans la section 1.1, nous commençons par donner des définitions générales du Linked Data ainsi que quelques statistiques concernant les données publiées. Ensuite, nous discutons dans la section 1.2 les défis scientifiques en s'appuyant sur l'état de l'art. Enfin, nous présentons dans la section 1.4 nos contributions permettant d'atteindre les différents objectifs de cette thèse.

Chapitre 2 : Technical background and related work

Ce chapitre est consacré à positionner les contributions proposées dans cette thèse par rapport à l'état de l'art. Nous commençons par une introduction du Web sémantique en section 2.1 avec un rappel de la syntaxe et la sémantique du modèle de données RDF. Nous donnons un aperçu du langage de requête SPARQL 1.1 RDF, et comment il peut être utilisé pour traiter les données RDF. Nous définissons l'inférence, notamment les inférences RDFS, qui permet la déduction des nouvelles données à partir de celles qui sont explicitement définies. Dans la section 2.2, nous donnons un aperçu des modèles de contrôle d'accès les plus connus dans la littérature, suivi par les stratégies de résolution des conflits utilisées pour résoudre les conflits qui découlent de l'utilisation des autorisations négatives. Ensuite, dans la section 2.3, nous présentons les travaux qui ont été proposés pour contrôler l'accès aux données RDF, et les critères utilisés pour les comparer, suivis par la synthèse de l'étude de ces travaux en section 2.4. En se basant sur les critères qui ne sont pas bien considérés, nous présentons enfin dans la section 2.5 les solutions que nous avons proposées.

Chapitre 3 : A fine-grained access control model for RDF stores

Ce chapitre présente notre première contribution, à savoir un modèle de contrôle d'accès a grains fins pour les données RDF. La sémantique de notre modèle est présentée dans la section 3.1, dans laquelle nous commençons par donner la sémantique des autorisations qui est basée sur des motifs SPARQL, suivie par la définition des politiques de contrôle d'accès. Nous donnons des conditions qui doivent être respectées pour que la politique soit bien formée. Pour résoudre les conflits éventuels, nous donnons les caractéristique de la fonction abstraite de résolution de conflits, et nous définissons des propriétés qui permettent de prouver si une fonction applique une stratégie de résolution de conflits donnée. Enfin nous montrons dans la section 3.2 comment construire les stratégies de résolution de conflits connues dans la litérature de contrrôle d'accès.

Chapitre 4 : Inference leakage problem and solution

Dans ce chapitre, nous présentons le problème de fuite d'inférence et la solution proposée pour y faire face. La section 4.1 est dédiée à l'introduction du problème d'inférence dans les autres modèles de données. Par la suite, nous donnons, dans la section 4.2, nous donnons un exemple qui illustre le problème de fuite d'inférence suivi par la définition formelle de la propriété de cohérence. Nous proposons ensuite une solution à ce problème en section 4.3 dans laquelle nous proposons un algorithme de vérification statique qui permet de vérifier si une politique est cohérente Nous donnons les preuves de la correction et la complétude de notre algorithme. Enfin nous montrons par des exemples comment utiliser les contre-exemples générés par l'algorithme pour corriger la politique. Enfin nous montrons la deuxième alternative pour utiliser les graphes contre-exemples, qui consiste à utiliser les graphes contre-exemples comme contraintes d'intégrité.

Chapitre 5 : Policy administration

Dans ce chapitre nous présentons un langage de haut niveau basé sur les attributs, qui permet la définition de politiques multi-utilisateurs. Nous commençons par un exemple qui illustre l'idée de la représentation de la politique sous la forme d'un arbre. Ensuite, dans la section 5.1, nous donnons l'architecture globale de la solution proposée. Pour définir formellement la sémantique de notre langage, nous fournissons une syntaxe de notre formalisme par une grammaire BNF étendue (EBNF) en section 5.2. Nous présentons ensuite la sémantique de chaque composant du langage. Dans la section 5.3, nous montrons par des exemples, la façon avec laquelle, une demande d'un sujet est évaluée sur une politique globale pour générer une politique utilisateur. Enfin, nous donnons un exemple complet de l'évaluation d'une requête d'un utilisateur ayant un certain nombre d'attributs.

Chapitre 6 : Policy enforcement and experiments

Dans ce chapitre nous montrons comment appliquer des politiques multi-utilisateurs en utilisant une approche d'annotation de données. Nous commençons par présenter notre méthode d'encodage dans la section 6.1. Nous prouvons que notre encodage est correct et montrons comment une requête utilisateur est évaluée sur une graphe annoté. Nous donnons ensuite dans la section 6.2, des détails de l'implémentation de notre solution sur un entrepôt RDF concret, à savoir Jena TDB. A la fin, nous présentons les résultats de nos expérimentations qui sont séparés en deux parties : statique et dynamique. La partie statique présente les temps d'annotation du graphe de base en variant la taille du graphe de base et le nombre d'autorisations. La partie dynamique présente les temps d'évaluation des requêtes en variant la taille du graphe de base,le nombre d'autorisations de la politique et enfin les requêtes et les autorisations assignées à l'utilisateur.

Chapitre 7 : Conclusion

Ce chapitre rappelle les contributions et présente des extensions et des travaux futurs. Ces derniers sont subdivisés par rapport aux exigences sur lesquelles nos propositions ont été conduites, à savoir : l'expressivité, la vérifiabilité et la performance.

Références

[ABF$^+$04] Rakesh Agrawal, Roberto Bayardo, Christos Faloutsos, Jerry Kiernan, Ralf Rantzau, and Ramakrishnan Srikant. Auditing compliance with a hippocratic database. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 516–527. VLDB Endowment, 2004.

[ADCH$^+$07] Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in RDF stores. In *The Semantic Web*, pages 1–14. Springer, 2007.

[BL06] Tim Berners-Lee. Linked data-design issues. 2006. `https://www.w3.org/DesignIssues/LinkedData.html`.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[FFMA10] Giorgos Flouris, Irini Fundulaki, Maria Michou, and Grigoris Antoniou. Controlling access to rdf graphs. In *Future Internet-FIS 2010*, pages 107–117. Springer, 2010.

[FJ02] Csilla Farkas and Sushil Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 4(2):6–11, 2002.

[GW76] Patricia P. Griffiths and Bradford W. Wade. An authorization mechanism for a relational database system. *ACM Trans. Database Syst.*, 1(3):242–255, 1976.

[HPS14] Patrick J Hayes and Peter F Patel-Schneider. Rdf 1.1 semantics. *W3C Recommendation*, 2014. `http://www.w3.org/TR/rdf11-mt/`.

[JF06] Amit Jain and Csilla Farkas. Secure resource description framework: an access control model. In *SACMAT*, pages 121–129. ACM, 2006.

[KSR$^+$09] Georgi Kobilarov, Tom Scott, Yves Raimond, Silver Oliver, Chris Sizemore, Michael Smethurst, Christian Bizer, and Robert Lee. Media meets semantic web - how the BBC uses dbpedia and linked data to make connections. In *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, pages 723–737, 2009.

[Lam81] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[LKZ$^+$12] Nuno Lopes, Sabrina Kirrane, Antoine Zimmermann, Axel Polleres, and Alessandra Mileo. A logic programming approach for access control over RDF. In *ICLP 2012, Hungary*, pages 381–392, 2012.

[PMF$^+$12] Vassilis Papakonstantinou, Maria Michou, Irini Fundulaki, Giorgos Flouris, and Grigoris Antoniou. Access control for RDF graphs using abstract models. In *SACMAT*, pages 103–112, 2012.

[RFJ05] Pavan Reddivari, Tim Finin, and Anupam Joshi. Policy-based access control for an RDF store. In *WWW*, pages 78–81, 2005.

[RKKT14] Jyothsna Rachapalli, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani Thuraisingham. Towards fine grained RDF access control. In *SACMAT*, pages 165–176. ACM, 2014.

[RSA78] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

Le manuscrit en anglais

Abstract

The emergence of the Semantic Web has led to a rapid adoption of the RDF (Resource Description Framework) to describe the data and the links between them. The RDF graph model is tailored for the representation of semantic relations between Web objects that are identified by IRIs (Internationalized Resource Identifier). The applications that publish and exchange potentially sensitive RDF data are increasing in many areas: bioinformatics, e-government, open data movement. The problem of controlling access to RDF content and selective exposure to information based on privileges of the requester becomes increasingly important. Our main objective is to encourage businesses and organizations worldwide to publish their RDF data into the linked data global space. Indeed, the published data may be sensitive, and consequently, data providers may avoid to release their information, unless they are certain that the desired access rights of different accessing entities are enforced properly, to their data. Hence the issue of securing RDF content and ensuring the selective disclosure of information to different classes of users is becoming all the more important. In this thsesis, we focused on the design of a relevant access control for RDF data. The problem of providing access controls to RDF data has attracted considerable attention of both the security and the database community in recent years. New issues are raised by the introduction of the deduction mechanisms for RDF data (e.g., RDF/S, OWL), including the *inference leakage problem*. Indeed, when an owner wishes to prohibit access to information, she/he must also ensure that the information supposed secret, can not be inferred through inference mechanisms on RDF data.

In this PhD thesis we propose a fine-grained access control model for RDF data. We illustrate the expressiveness of the access control model with several conflict resolution strategies including *most specific takes precedence*. To tackle the inference leakage problem, we propose a static verification algorithm and show that it is possible to check in advance whether such a problem will arise. Moreover, we show how to use the answer of the algorithm for diagnosis purposes. To handle the subjects' privileges, we define the syntax and semantics of a XACML inspired language based on the subjects' attributes to allow much finer access control policies. Finally, we propose a data-annotation approach to enforce our access control model, and show that our solution incurs reasonable overhead with respect to the optimal solution which consists in materializing the user's accessible subgraph.

Keywords: RDF, Authorization, Semantic Reasoning, Inference Leakage, Enforcement, Linked Data

Résumé

L'émergence du Web sémantique a mené à une adoption rapide du format RDF (Resource Description Framework) pour décrire les données et les liens entre elles. Ce modèle de graphe est adapté à la représentation des liens sémantiques entre les objets du Web qui sont identifiés par des IRI. Les applications qui publient et échangent des données RDF potentiellement sensibles augmentent dans de nombreux domaines : bio-informatique, e-gouvernement, mouvements open-data. La problématique du contrôle des accès aux contenus RDF et de l'exposition sélective de l'information en fonction des privilèges des requérants devient de plus en plus importante. Notre principal objectif est d'encourager les entreprises et les organisations à publier leurs données RDF dans l'espace global des données liées. En effet, les données publiées peuvent être sensibles, et par conséquent, les fournisseurs de données peuvent être réticents à publier leurs informations, à moins qu'ils ne soient certains que les droits d'accès à leurs données par les différents requérants sont appliqués correctement. D'où l'importance de la sécurisation des contenus RDF est de l'exposition selective de l'information pour différentes classes d'utilisateurs. Dans cette thèse, nous nous sommes intéressés à la conception d'un contrôle d'accès pertinents pour les données RDF. De nouvelles problématiques sont posées par l'introduction des mécanismes de déduction pour les données RDF (e.g., RDF/S, OWL), notamment le problème de fuite d'inférence. En effet, quand un propriétaire souhaite interdire l'accès à une information, il faut également qu'il soit sûr que les données diffusées ne pourront pas permettre de déduire des informations secrètes par l'intermédiaire des mécanismes d'inférence sur des données RDF.

Dans cette thèse, nous proposons un modèle de contrôle d'accès à grains fins pour les données RDF. Nous illustrons l'expressivité du modèle de contrôle d'accès avec plusieurs stratégies de résolution de conflits, y compris la *Most Specific Takes Precedence*. Nous proposons un algorithme de vérification statique et nous montrons qu'il est possible de vérifier à l'avance si une politique présente un problème de fuite d'inférence. De plus, nous montrons comment utiliser la réponse de l'algorithme à des fins de diagnostics. Pour traiter les privilèges des sujets, nous définissons la syntaxe et la sémantique d'un langage inspiré de XACML, basé sur les attributs des sujets pour permettre la définition de politiques de contrôle d'accès beaucoup plus fines. Enfin, nous proposons une approche d'annotation de données pour appliquer notre modèle de contrôle d'accès, et nous montrons que notre implémentation entraîne un surcoût raisonnable durant l'exécution.

Mots-clés : RDF, Autorisation, Raisonnement sémantique, Fuite d'inférence, Application, Linked Data

Contents

Chapter 1

Introduction

Contents

The Web is becoming the main information dissemination for private and public organizations. A huge amount of information is exchanged every day. The traditional web is used to transmit, receive, and display content encapsulated as documents. These documents were intended for human consumption and could not be understood by machines. *The Semantic Web* also referred to as *web of data* can be conceptualized as an extension of the current Web so as to enable the creation, sharing and intelligent re-use of machine-readable content on the Web.

1.1 Context

Linked Data [Bizer 2009] is simply about using the Web to create typed links between data from different sources. Technically, Linked Data refers to a set of best practices for publishing and connecting structured data on the Web in such a way that they are machine-readable, their meaning is explicitly defined, they are linked to other external data sets, and can in turn be linked to from external data sets [Berners-Lee 2006]. Berners-Lee discusses the four basic principles for linked data as:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things

The Linked (Open) Data trend [1] is followed by institutions (eg., Higher Education and Research in France [2]), local authorities (eg., Cities of Lyon and Grenoble [3]), Thematic databases (eg., Dbtune, MusicBrainz [4]), Organizations (BBC in its internal content production systems [Kobilarov 2009]), geographical and of course general data (eg., GeoNames, Wikipedia [5])).

Linking data distributed across the Web requires a standard mechanism for specifying the existence and meaning of connections between items described in this data. This mechanism is provided by the Resource Description Framework (RDF), which is examined in detail in Chapter 2. Figure 1.1 shows datasets that have been published in Linked Data format [6]. The figure shows datastores that belong to different thematic domains (government, publications, life sciences, etc). The size of the Web of Data is estimated to over 3400 open sources for a total volume of over 85 billion triples (statements) [7], and more than 8500 sources on the open data portal of the European Union [8]

RDF has gained a lot of attention, and as a result an increasing number of data sets are now being represented with this language. From this popularity stemmed the need to efficiently store, query and reason over large amounts of

[1] http://www.mckinsey.com/business-functions/business-technology/our-insights/open-data-unlocking-innovation-and-performance-with-liquid-information

[2] http://data.enseignementsup-recherche.gouv.fr

[3] http://data.grandlyon.com and http://data.beta.metropolegrenoble.fr

[4] http://dbtune.org and http://linkedbrainz.org

[5] http://www.geonames.org/ontology and wiki.dbpedia.org

[6] http://lod-cloud.net/

[7] https://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets/LinkStatistics

[8] https://open-data.europa.eu/fr/data

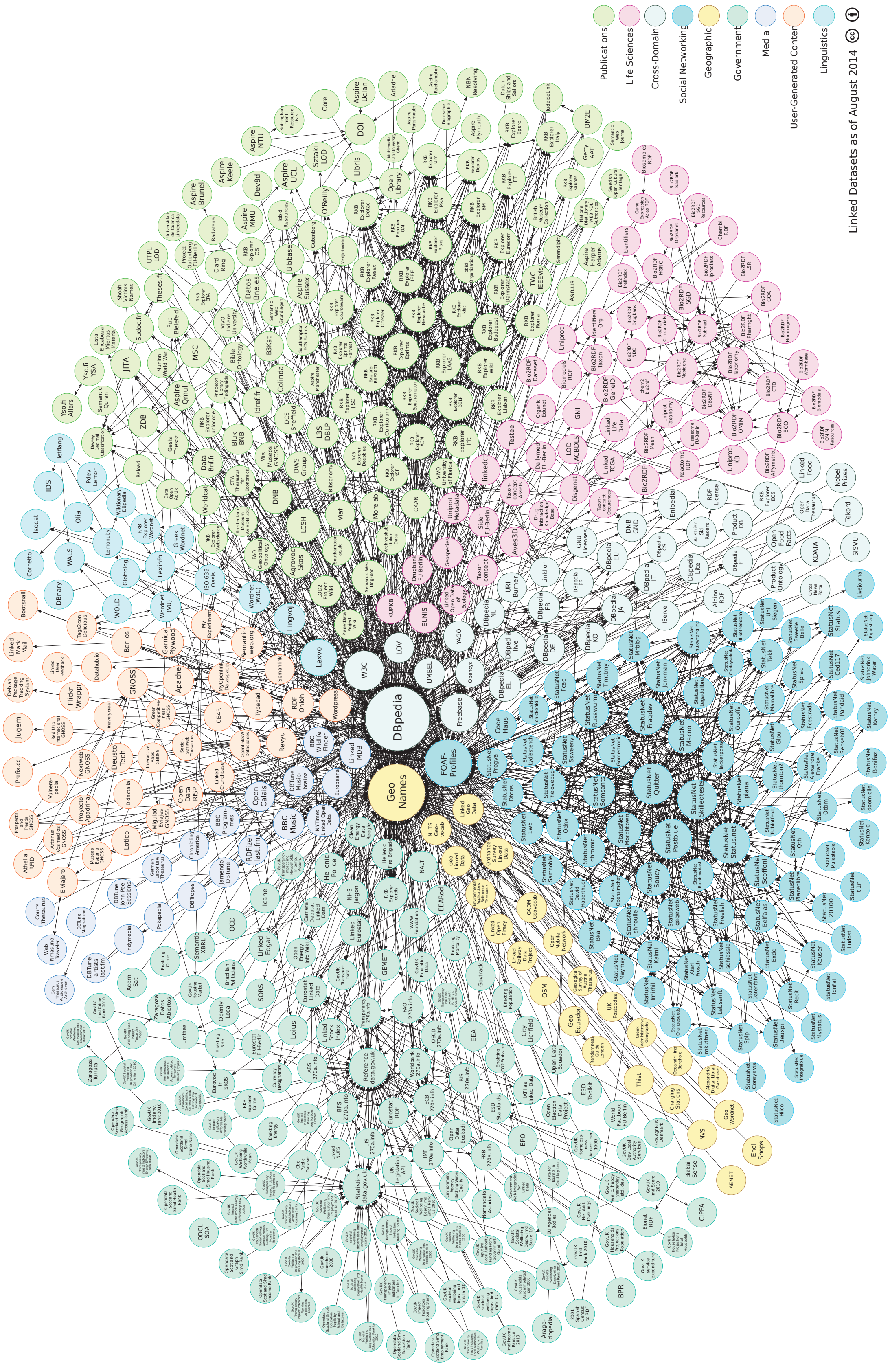


Figure 1.1: Linked Data Diagram (30/08/2014)

RDF data which has obviously motivated some research work on the design of adapted and efficient database systems. These systems are frequently referred to as *RDF stores* or *triplestores*. Several RDF stores have been developed to handle RDF data, among which some are native whereas others use RDBMS backend. Native RDF stores are those that are implemented from scratch and exploit the RDF data model to efficiently store and access the RDF data, such as 4Store [9], AllegroGraph [10], StarDog [11], Jena TDB [12] and Sesame Native Store [13]. RDMS-backed RDF stores such as Virtuoso [14] and Jena SDB [15], use a relational database to store triples. One of the key concepts of the Semantic Web architecture are *Named Graphs* which is a simple extension of the RDF data model that transforms RDF triples to *quads*. Named graphs are sets of triples identified by IRIs, allowing descriptions to be made of that set such as provenance information, context and other metadata.

As the demand for data and information management increases, there is also a critical need for maintaining the security of the data sources, applications, and information systems. Different fields of study have been introduced to design appropriate approaches able to provide security guarantees. These security fields involve authentication [Lamport 1981], access control [Griffiths 1976], cryptography [Diffie 1976, Rivest 1978] and auditing [Agrawal 2004]. Following the evolution of these fields, database systems offer new security paradigms to cope with vulnerabilities induced by new features (e.g., distribution, heterogeneity, autonomy, reasoning).

The protection of resources against unauthorized accesses, is one of the main features of todays systems. One of the main security services needed to achieve data protection is access control. Access control ensures that a user can access only resources she/he is allowed to see. Historically, the first access control model was defined by [Griffiths 1976] in the framework of the System R DBMS. In this model, the protected objects are tables and views, and the possible access modes that subjects can exercise on tables correspond to SQL operations that can be executed on tables. Later, other access control models have been proposed for different data models such as object-oriented DBMS [Rabitti 1991] and XML [Damiani 2002].

In this thesis, we focus on the security challenges that mainly arise in Linked Data context, particularly the selective disclosure of RDF data.

[9] www.4store.com
[10] http://franz.com/agraph/allegrograph/
[11] http://www.stardog.com/
[12] https://jena.apache.org/documentation/tdb/index.html
[13] http://www.rdf4j.org/
[14] http://virtuoso.openlinksw.com/
[15] https://jena.apache.org/documentation/sdb/

Our main objective is to encourage businesses and organizations worldwide to publish their RDF data into the linked data global space. Indeed, the published data may be sensitive, and consequently, data providers may avoid to release their information, unless they are certain that the desired access rights of different accessing entities are enforced properly, to their data, and that no sensitive data are revealed (by mistake). Hence the issue of securing RDF content and ensuring the selective disclosure of information to different classes of users is becoming all the more important.

An inspiring quote from Paul Terry who wrote in CTO Vision [16] : *To realize value from your data, you need to be able to share it among many stakeholders-internal lines of business, partners, researchers, and many others. But you also have a moral, ethical, and often legal, obligation to make sure that data is used responsibly. That means protecting individuals' privacy and assuring that their data is used only for legitimate purposes. As you gather more data from more parts of your organization, that gets very tricky very fast. [...] It quickly becomes clear that this is about more than simply checking a box for whether data is "secure". It's incredibly valuable to have all that data in one place where it can be analyzed, but you need to assure that different types of stakeholders can see only the information they legitimately need, and no more.*

1.2 Problem statement

In this thesis, we are interested in RDF data disclosure and inference leakage problem. The problem of providing access controls to RDF data has attracted considerable attention of both the security and the database community in recent years. Our goal is to define an authorization policy to be integrated to RDF stores in order to control RDF data disclosure. Moreover, we must ensure that the disclosed data can not be used to infer confidential information. In this context, we are interested in the following issues :

1.2.1 Selective RDF data disclosure

Given the sensitive nature of information, different portions of RDF data may require different access rights with respect to the privileges of the requester. In this thesis, we are interested in selectively disclosing information based on the

[16] https://ctovision.com/2015/12/big-data-unlocks-valuable-information-across-organizations-but-only-if-you-can-protect-it/

RDF content, who requests it and under what context. Most of nowadays RDF Stores offer protection at named graph level. For instance, such a feature has been introduced in 4STORE version 1.1.5[17]. Graph-based security is available in VIRTUOSO[18] or in STARDOG version 3.1[19] as well. The issue with these models is that the access control policies are defined over named graphs which must be created with respect to policies. For instance if a policy states that the nurses have access to patient's records, then all triples related to the patient's records must be gathered into one named graph over which the policy is defined. Moreover, complex policies may lead to the creation of several named graphs. Having a large number of complex policies may lead the administrator to create several named graphs that may be difficult to manage. In addition, a redundancy problem arises because a triple may belong to several named graphs. Some RDF stores such as ALLEGROGRAPH[20] support triple pattern based access control[21] which allows the definition of simple authorizations such as *deny or allow access to triples representing patients' records*. However, more expressive policies can not be specified. For instance an authorization such as *Deny access to patients' records if they have cancer* cannot be specified.

Given the open nature of the web where the RDF data are published, the subject may not be known by the system prior to the submission of a query. Hence, the selective disclosure can not only rely on traditional identity based and role based access control policies. Attribute Based Access Control (ABAC) model gives more flexibility by defining authorizations on the basis of subject's and object's attributes. ABAC avoids the need for explicit authorizations to be directly assigned to individual subjects prior to a request to perform an operation on the object.

1.2.2 Inference leakage

The second issue is to ensure that sensitive information can not be inferred once the data have been disclosed to the user. This problem is known in the access control literature as the *Inference problem* [Farkas 2002] called in this manuscript the *Inference leakage problem*. According to the 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, automati-*

[17] http://4store.org/trac/wiki/GraphAccessControl

[18] http://docs.openlinksw.com/virtuoso/rdfgraphsecurity.html

[19] http://docs.stardog.com/#_security

[20] http://franz.com/agraph/allegrograph

[21] http://franz.com/agraph/support/documentation/v4/security.html#filters

cally analyzing the content of the data". Inference rules are used to derive new triples from those explicitly asserted in an RDF store. In particular, a set of inference rules known as RDF *Schema* (RDFS) is standardized [Hayes 2014]. Authorization models for RDF data have been proposed to control accesses to RDF data, both in the presence of inference rules [Reddivari 2005, Lopes 2012, Papakonstantinou 2012, Jain 2006] or not [Abel 2007, Flouris 2010, Rachapalli 2014]. However, the issue is that inference capabilities can be used by a malicious user to infer sensitive information from authorized ones.

To illustrate the 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 :`hasTumor` predicate is :`Cancerous` (e.g. (:`hasTumor` ;`rdfs` :`domain` ;:`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 the deduction system is enriched with user-defined rules.

1.2.3 Enforcement and performance

Specific issues arise from the enforcement of the access control. The first issue is the impact of access control enforcement on the system's performance : we must ensure that the enforcement of the access control incurs a low overhead in the RDF store performance. The second issue regards the mechanisms needed to enforce the access control. The enforcement mechanism must be deployable in the RDF store with minimal additional mechanisms and ideally, with no alteration of its internal components

1.3 Related work study

A first study of the state of the art has led to the definition of the evaluation criteria in order to compare the related works in access control to RDF data, including the expressiveness of policy specification languages, conflict resolution generated by conflicting decisions and the verification of unauthorized inferences. The study has resulted in the determination of well treated criteria such as the supported actions, the expressiveness of objects and the protection of explicit and implicit triples. Other criteria have not been deeply considered,

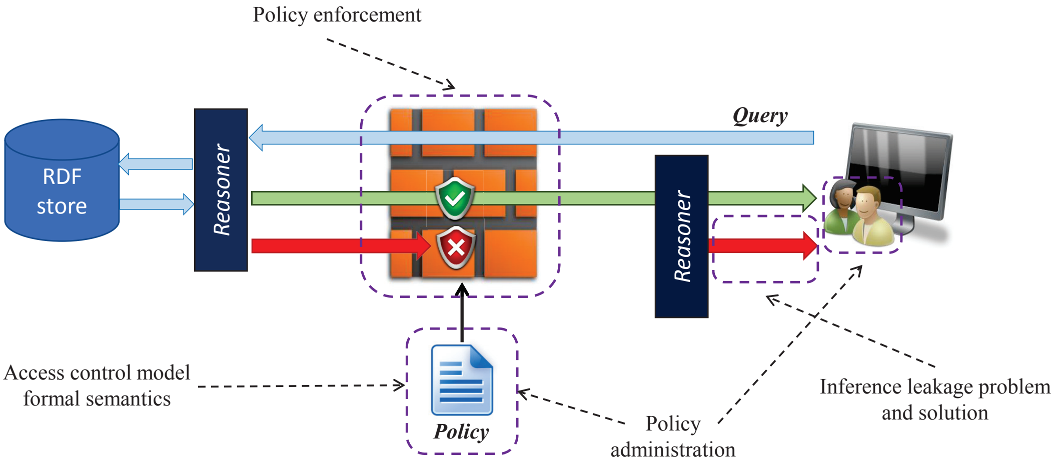

Policy enforcement
Query
RDF store
Reasoner
Reasoner
Access control model formal semantics
Policy
Policy administration
Inference leakage problem and solution

Figure 1.2: Main contributions

including conflict resolution, the expressiveness of the subjects and the inference leakage. In Chapter 2 we introduce the Semantic Web and a recalling of the syntax and semantics of the RDF Data Model. We give an overview about SPARQL 1.1 RDF query language, and how it can be used to process RDF data. We define inference, in particular RDFS inferencing, which allows the deduction of new data from those explicitly defined. We give an overview of the most known access control models found in the literature, followed by the conflict resolution strategies used to resolve conflicts that stem from the use of negative authorizations. Next, we present the works that were proposed to control access to RDF data, and the criteria used to compare them. We finally present the criteria that have not been well considered, which are the basis of our proposals.

1.4 Contributions

We present in the following, the main contributions of this thesis depicted by Figure 1.2.

1.4.1 Access control model for RDF

We define in Chapter 3 the formal semantics of a triple-level access control model for RDF. The semantics are defined by means of *positive (authorized) subgraph* from the base graph, over which the user's query is evaluated. This makes the model independent from the query language. Our model supports negative authorizations to handle real-life exceptions. This leads to possible conflicts that may occur between authorizations. Conflict resolution strategies are used to resolve such conflicts by selecting the preferred authorizations to apply, with respect to some properties of the authorizations. Whereas most of the works hard-code their strategies, others define them as parameters that are fixed by the administrator during policy design. We propose a more liberal approach by defining our policy using an *abstract conflict resolution function* which is defined by the administrator. We illustrate the applicability of the authorization model by showing that usual conflict resolution strategies can be expressed in our framework. We show how to build classic conflict resolution strategies namely : *Denials Take Precedence*, *Permissions Take Precedence*. Moreover, we show how to build more elaborate strategies such as *Most Specific Takes Precedence* to handle exceptions.

1.4.2 Inference leakage problem and solution

To deal with the inference leakage problem, we propose in Chapter 4 an 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 sensitive information disclosure. Several authorization models for RDF which consider inference use annotations to determine whether the inferred triples are accessible or not [Reddivari 2005, Lopes 2012, Papakonstantinou 2012]. 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 [Jain 2006], but the base RDF graph kept in the RDF store is needed and conflict resolution strategies are hard-coded in their algorithm. We identify and formalize a *consistency property* that captures the inference 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. This property ensures that confidential information can not be inferred from authorized information with respect to a set of inference rules. To solve the issue, 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 and it is constructive: whenever the answer is positive, a set of counterexample graphs is computed. This answer can be used by the administrator to analyze and then solve the issue. We show how the counterexamples could be used by the administrator to fix the policy, or how to use them as integrity constraints that do not allow updates which could lead to inference leakage

1.4.3 Policy administration

The policies of the model proposed in Chapter 3 are defined without specifying the subject for which permissions are assigned. This allows to use any upstream access control model to map the users to their assigned permissions. We propose in Chapter 5 a high level access control language which permits the definition of global policies which are then compiled into subject specific policies enforced by our access control model. We have chosen to define our policies on the basis of the user attributes following the Attribute Based Access Control Approach (ABAC), where access to protected resources is based on users having specific attributes (eg. name, role, date of birth, address, phone number, etc.). This approach allows a much finer approach of access control combining not only user attributes, but other environmental information, such as location and time. Rather than just using the role of a user to decide whether or not to grant access to protected resource, ABAC combines multiple attributes to make a context-aware decision regarding individual requests for access. ABAC is implemented through XACML which is a declarative language to define policies which are structured as a tree of sub-policies. We define the syntax and semantics of a language inspired by XACML by defining the main components of the proposed solution and showing how the user's policy is created and enforced. Intuitively, a global policy can be represented in a *Tree structure* where the intermediate nodes represent policies, the leaves are authorizations and nodes edges are labeled with *targets* representing attribute based conditions. The latter are evaluated using key/-value pairs representing attributes provided by subjects. The subject provides her/his attributes in a request which is evaluated over the global policy tree to determine her/his assigned authorizations. Based on these authorizations, the subject's query is evaluated over her/his positive subgraph returning the her/his accessible triples.

1.4.4 Policy enforcement

In Chapter 6, we propose an enforcement framework based on data-annotations for our access control model. We propose an approach where we annotate every triple of the base graph with a bitset representing its applicable authorizations. Similarly, users are assigned bitsets representing the authorizations applicable to them. We show the annotation process of the base graph, and how the user query is evaluated over the annotated graph, and we prove that our encoding is correct. We use the graph name position to store the bitset of applicable authorization of the triple. As todays RDF stores support named graphs [Haslhofer 2011], no additional mechanisms are needed to enforce our model, in contrast to approaches that use specific extensions of RDF data model to store annotations. The annotation process is performed by transforming the authorizations into (`CONSTRUCT`) queries that are evaluated using the RDF store query engine. Once the base graph annotated, it is made available to the requesters. When a query is sent from a requester, her/his positive subgraph is computed by making *logical and* between the requester's bitset, and the triple's bitset. The result is used to determine the triple's that take part of the positive subgraph. The requester's query is then evaluated over her/his positive subgraph and the result is returned to the user. Finally we show the results of experiments of our solution implemented on a concrete RDF store. We show that our implementation incurs *reasonable overhead* at runtime (about +50%) with respect to the optimal solution which consists in materializing the user's accessible subgraph.

Chapter 2

Technical background and related work

Contents

▷ *In this chapter, we start by introducing the Semantic Web (aka Web of Data) and recalling the syntax and semantics of the RDF Data Model. We present an overview of the RDF query language SPARQL, and how it can be used to process RDF data. We present* inference, *in particular RDFS inferencing, a key feature of the Semantic Web, which allows the deduction of new data from those explicitly defined. We give an overview of the most known access control models found in the literature, followed by the conflict resolution strategies used to resolve conflicts that stem from the use of negative authorizations. We present the works that have been proposed to control access to RDF data, and the criteria used to compare them. We finally give the summary of our study by showing the criteria that have not been well considered, which are the basis of our proposals.* ◁

2.1 Semantic Web

Most of today's Web content is suitable for human consumption. Even Web contents that are generated automatically from databases are usually presented without the original structural information found in databases. The Semantic Web initiative consists in representing Web contents in a form that is more easily machine-processable. World Wide Web Consortium (W3C) is promoting the mobilization of semantic web technology. In their vision, semantic web is achieved with a layered stack solution (Figure 2.1). RDF(S) and OWL specifications have been suggested as mature recommendations.

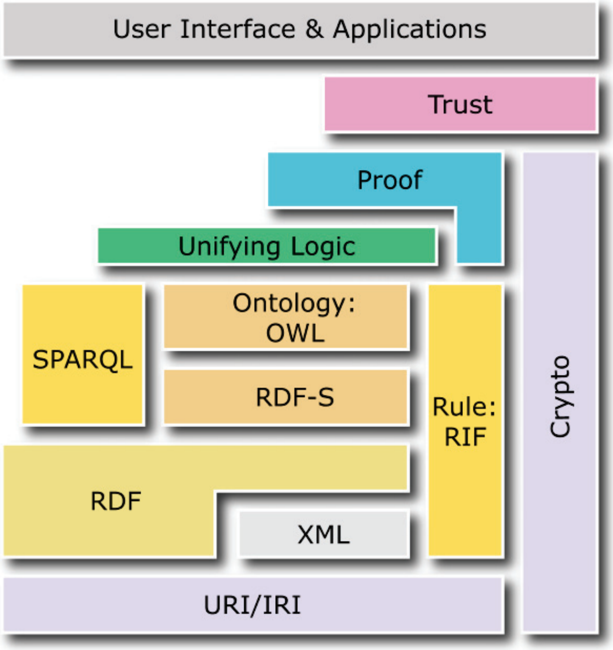


Figure 2.1: Semantic Web Layers

2.1.1 Graph Data Model

Graph database models can be defined as those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors [Angles 2008]. The graph data model used in the semantic web is RDF (*Resource Description Framework*) [Hayes 2014].

2.1.1.1 Resource Description Framework

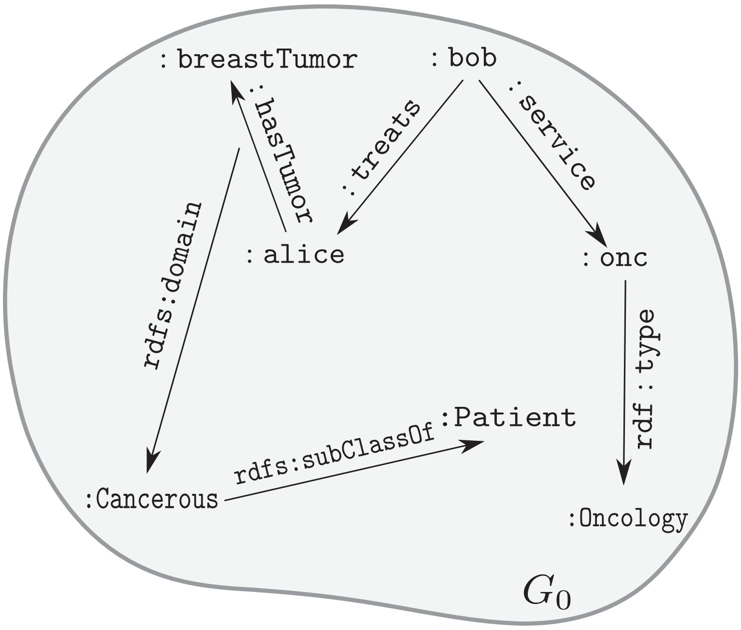
RDF has been developed under the auspices of the W3C and has become the de facto standard for representing semantic relations between web resources which are uniquely identified by *Universal Resource Identifiers* (URIs) or *Internationalized Resource Identifiers* (IRIs) [Cyganiak 2014]. URIs represent common global identifiers for resources across the Web. An IRI is an extension of a URI which allows the use of Unicode characters. The syntax and format of IRIs is very similar to the well-known uniform resource locators (URLs); e.g., `http://example.com/hospital#alice`. In fact, URLs are just special cases of IRIs. Another form of IRIs is a Uniform Resource Name (URN), which identifies something that is not associated to a Web resource but on which people on the Web want to write about such as books. To represent value data types such as numbers, booleans and strings, RDF uses *Literals*. In RDF, a resource which do not have an IRI can be identified using *Blank nodes*. Blank nodes are used to represent these unknown resources, and also used when the relationship between a subject node and an object node is n-ary (as is the case with collections). The purpose of RDF is to promote encoding, exchanging and reusage of structured metadata. RDF allows to decompose knowledge into small portions called *triples* or *statements*. Triples have the form "(`subject` ; `predicate` ; `object`)" built from pairwise disjoint countably infinite sets I, B, and L for IRIs, blank nodes, and literals respectively. The subject represents the resource for which information is stored and is identified by an IRI. The predicate is a property or a characteristic of the subject and is identified by an IRI. The object is the value of the property and is represented by an IRI of another resource or a literal.

For ease of notation, in RDF, one may define a *prefix* to represent a namespace, such as `rdf : type` where `rdf` represents the namespace `http://www.w3.org/1999/02/22-rdf-syntax-ns`

Note 2.1.1 *In this manuscript, 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.*

For instance the triple (`:alice` ; `:hasTumor` ; `:breastTumor`) states that alice has a breast tumor. A collection of RDF triples is called an *RDF Graph* and can be intuitively understood as a directed labeled graph where resources represent the nodes and the predicates the arcs as shown in Figure 2.2.

Definition 2.1.2 (RDF graph) *An RDF graph (or simply "graph", where unambiguous) is a finite set of RDF triples.*



	Subject	Predicate	Object
et_1	:hasTumor	rdfs :domain	:Cancerous
et_2	:Cancerous	rdfs :subClassOf	:Patient
et_3	:onc	rdf :type	:Oncology
et_4	:alice	:hasTumor	:breastTumor
et_5	:bob	:service	:onc
et_6	:bob	:treats	:alice

Figure 2.2: An example of an RDF graph G_0

Example 2.1.3 *Figure 2.2 depicts a graph G_0 constituted by triples et_1 to et_6, both pictorially and textually.*

In this manuscript, we reuse the formal definitions and notation used by Pérez and Gutierrez [Pérez 2009]. Throughout the manuscript, $\mathscr{P}(\mathsf{E})$ denotes the *finite powerset* of a set E and $\mathsf{F} \subseteq \mathsf{E}$ denotes a *finite subset* F.

RDF Semantics The W3C document [Hayes 2014] defines a model-theoretic semantics for RDF 1.1. Model-theoretic semantics for a language assumes that the language refers to a *world*, and describes the minimal conditions that a world must satisfy in order to assign an appropriate meaning for every expression in the language. A particular world is called an *interpretation*. An RDF interpretation is a mapping from IRIs and literals into a set, together with some constraints upon the set and the mapping.

Definition 2.1.4 (Simple interpretation) *A simple interpretation $\mathscr{I}$ of a given vocabulary V consists of:*

1. *A non-empty set* IR *of resources, called the domain or universe of* $\mathscr{I}$.
2. *A set* IP*, called the set of properties of* $\mathscr{I}$.
3. $\mathsf{IEXT} : \mathsf{IP} \rightarrow \mathscr{P}(\mathsf{IR} \times \mathsf{IR})$ *a mapping from properties into set of pairs in* IR.
4. $\mathsf{IS} : V \rightarrow (\mathsf{IR} \cup \mathsf{IP})$ *a mapping from IRIs in V into the union of sets* IR *and* IP.
5. *A partial mapping* IL *from literals into* IR

Given a triple (s, p, o), $\mathscr{I}((s, p, o)) = true$ if $s, p, o \in V$, $\mathsf{IS}(p) \in \mathsf{IP}$ and $(\mathsf{IS}(s), \mathsf{IS}(o)) \in \mathsf{IEXT}$; otherwise, $\mathscr{I}((s, p, o)) = false$. Given a set of triples S, $\mathscr{I}(S) = false$ if $\mathscr{I}((s, p, o)) = false$ for some triple (s, p, o) in S, otherwise $\mathscr{I}(S) = true$. $\mathscr{I}$ satisfies S, written as $\mathscr{I} \models S$ if $\mathscr{I}(S) = true$; in this case, we say $\mathscr{I}$ is a simple interpretation of S.

Example 2.1.5 *Consider the vocabulary $V = \{:\texttt{bob}, :\texttt{service}, :\texttt{onc}, :\texttt{Oncology}, \texttt{rdf} :\texttt{type}\}$. The following is a simple interpretation $\mathscr{I}$ of V.*

$\mathsf{IR} = \{a, b, c\}$
$\mathsf{IP} = \{e, f\}$
$\mathsf{IEXT} = \{e \mapsto \{\langle a, b\rangle\}, f \mapsto \{\langle b, c\rangle\}\}$
$\mathsf{IS} = \{:\texttt{bob} \mapsto a, :\texttt{service} \mapsto e, :\texttt{onc} \mapsto b, :\texttt{Oncology} \mapsto c, \texttt{rdf} :\texttt{type} \mapsto f\}$

The verification of triples using $\mathscr{I}$ is done as follows:

$(\mathsf{IS}(:\texttt{bob}), \mathsf{IS}(:\texttt{onc})) = (a, b) \in \mathsf{IEXT}(e) = \mathsf{IS}(:\texttt{service})$.
Hence $\mathscr{I}(:\texttt{bob} \,; :\texttt{service} \,; :\texttt{onc}) = true$
$(\mathsf{IS}(:\texttt{onc}), \mathsf{IS}(:\texttt{Oncology})) = (b, c) \in \mathsf{IEXT}(f) = \mathsf{IS}(\texttt{rdf} :\texttt{type})$.
Hence $\mathscr{I}(:\texttt{onc} \,; \texttt{rdf} :\texttt{type} \,; :\texttt{Oncology}) = true$.
Thus $\mathscr{I}$ is a simple interpretation of S
Where $S = \{(:\texttt{bob} \,; :\texttt{service} \,; :\texttt{onc}), (:\texttt{onc} \,; \texttt{rdf} :\texttt{type} \,; :\texttt{Oncology})\}$

Simple interpretation has no particular extra conditions on the vocabulary. Additional extensions of simple interpretation have been introduced, such as *RDF(S)-Interpretation* which assign special meanings to the symbols in particular vocabularies (RDF(S)). For more details, we refer the reader to [Hayes 2014].

2.1.1.2 SPARQL

We just presented RDF as the data model of the Semantic Web. In general, a data model comes equipped with a language to support queries over some data set. The emergence of RDF recommendation has spun up several RDF query languages, such as SPARQL [Harris 2013], RQL [Lassner 2002], SeRQL [Broekstra 2003], TRIPLE [Sintek 2002], RDQL [Seaborne 2004] and N3 [Berners-Lee 2011]. An RDF Query Language is a formal language used for querying RDF Triples from an *RDF Store* also called *Triple Store*. An RDF store is a database specially designed for storing an retrieving RDF triples. SPARQL (SPARQL Protocol and RDF Query Language) is a W3C recommendation and has established itself as the de facto language for querying RDF data. SPARQL borrowed part of its syntax from the popular and widely adopted Structured Query Language (SQL). The main mechanism for computing query results in SPARQL is subgraph matching: RDF triples in both the queried RDF data and the query patterns are interpreted as nodes and edges of directed graphs, and the resulting query graph is matched to the data graph using variables. The SPARQL building blocks are *graph patterns* which are built from pairwise disjoint countably infinite sets I, V and L for IRIs, variables, and literals respectively.

Definition 2.1.6 (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* $\mathsf{BGP} = \mathscr{P}(\mathsf{TP})$*.*

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* $\mathsf{var}(B) = \{v \mid \exists tp \in B \wedge v \in \mathsf{var}(tp)\}$*.*

In this manuscript, we do not explicitly use blank nodes which are replaced by *variables*. Blank nodes of RDF are semantically equivalent to existentially quantified variables [Polleres 2007]. 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. Moreover, we use an extended version of RDF [ter Horst 2005] which allows variables in property position. When graph patterns are considered as instances stored in an RDF store, we simply call them *graphs*.

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 2.1.7 (Substitution Mappings) *A* substitution (mapping) η *is a partial function* $\eta : \mathsf{V} \to \mathsf{T}$*. 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* $\eta^\star : \mathsf{T} \to \mathsf{T}$ *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* $\eta : ?\mathtt{v} \mapsto \eta_2(\eta_1(?\mathtt{v}))$ *when defined.*

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 2.1.8 (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:*

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

Example 2.1.9 shows the evaluation of a triple pattern.

Example 2.1.9 *Let* B *be defined as* $B = \{(?\mathtt{p}\,;\mathtt{rdf:type}\,;\mathtt{:Patient})\}$*.* B *is a triple pattern with a variable in the subject position which will be used to match all triples of type Patient. The evaluation of* B *on the example graph* G_0 *of Figure 2.2 is* $[\![B]\!]_{G_0} = \{\eta\}$*, where* η *is defined as* $\eta : ?\mathtt{p} \mapsto \mathtt{:alice}$*.*

Example 2.1.10 shows the evaluation of a basic graph pattern.

Example 2.1.10 *Let* B *be defined as* $B = \{(?\mathtt{d}\,;\mathtt{:service}\,;?\mathtt{s}), (?\mathtt{d}\,;\mathtt{:treats}\,;?\mathtt{p})\}$*.* B *returns the doctors, their services and the patients they treat. The evaluation of* B *on the example graph* G_0 *of Figure 2.2 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 [Pérez 2009] for further details).

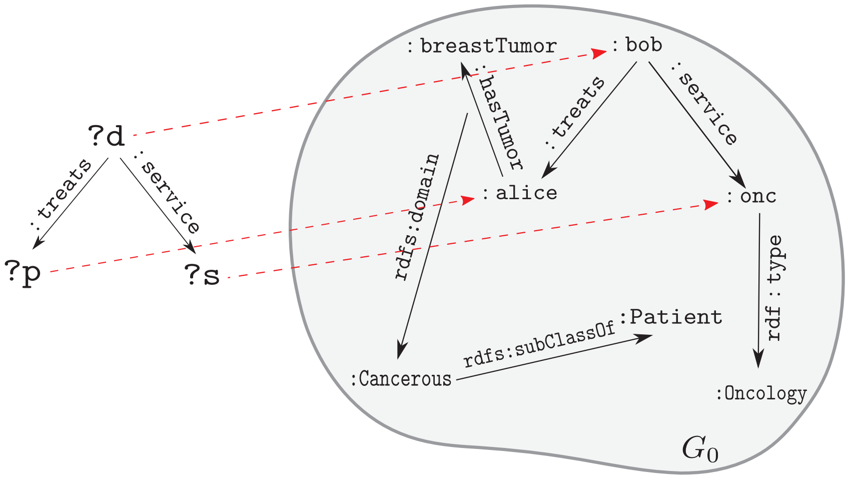


Figure 2.3: Graph Pattern evaluation

Another key concept of the Semantic Web is *named graphs* in which a set of RDF triples is identified using an IRI. This allows to represent meta-information about RDF data such as provenance information and context. In order to handle named graphs, SPARQL defines the concept of *dataset*. A dataset is a set composed of a distinguished graph called the *default graph* and pairs comprising an IRI and an RDF graph constituting named graphs.

Definition 2.1.11 (RDF dataset) *An RDF dataset is a set :*

$$\mathscr{D} = \{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$$

where $G_i \in \mathsf{BGP}$, $u_i \in \mathsf{I}$, and $n \geq 0$. In the dataset, G_0 is the default graph, and the pairs $\langle u_i, G_i\rangle$ are named graphs, with u_i the name of G_i.

SPARQL Query result forms SPARQL has four query result forms :

- `SELECT` returns all, or a subset of, the variables bound in a query pattern match
- `CONSTRUCT` returns an RDF graph constructed by substituting variables in a set of the BGP defined as template.
- `DESCRIBE` returns an RDF graph that describes the resources
- `ASK` returns a boolean indicating whether a query pattern matches or not an instance in the queried RDF graph.

2.1.1.3 Inference

One of the main features of the Semantic Web is the ability to do inferencing. Inference is the act of deriving new facts from known premises. A graph may contain *implicit triples* even though they are not explicitly present in it. *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.

An inference rule consists of a head and a body. The head is a triple pattern which represents the rule conclusion. The body is a BGP which represents the rule premises.

Definition 2.1.12 (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* $\mathbf{R}$*. The following notation is also used to represent inference rules:*

$$\frac{tp_1, \ldots, tp_k}{tp}$$

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. 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 [Abiteboul 1995, Chap. 12].

Definition 2.1.13 (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* $\mathsf{Cl}_{\mathbf{R}}(G) = G$

The above closure takes a graph and a set of inference rules and iteratively applies the rules over the union of the original graph and the inferences until a fixpoint. The following lemma shows that the closure is finite.

Lemma 2.1.14 (Finite Closure) *Let $\mathsf{Cl}_R(G) = \bigcup_{i \in \mathbb{N}} G_i$ the closure of a graph G according to a set of rules $\mathbf{R}$, there exists some $k \in \mathbb{N}$ such that for all $l \in \mathbb{N} . l \geq k \Rightarrow G_l = G_k$.*

Proof For the sake of the contradiction, assume that for all $i \in \mathbb{N}$, $G_i \subsetneq G_{i+1} = G_i \cup \phi_R(G_i)$. It follows that there exists some $x_i \in \phi_R(G_i)$ not in G_i obtained from some $r_i \in R$ and some $\sigma_i \in [\![\{tp_1^i, \ldots, tp_k^i\}]\!]_G$ with $\{tp_1^i, \ldots, tp_k^i\}$ the body of r_i. Repeating this construction for each i, we obtain a finite sequence of triples x_i with associated mapping σ_i. However, the condition $\mathsf{dom}(\eta) = \mathsf{var}(B) \wedge \eta(B) \subseteq G$ in Definition 2.1.8 ensures that only a *finite* number of mappings σ_i can be obtained: it is bounded by T^V where V is the total number of variables that appears in the heads of the rules from R and T is number of terms of G. This contradicts the hypothesis. ■

The following is an example of the application of an inference rule on a given graph.

Example 2.1.15 *Consider the following inference rule 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.*

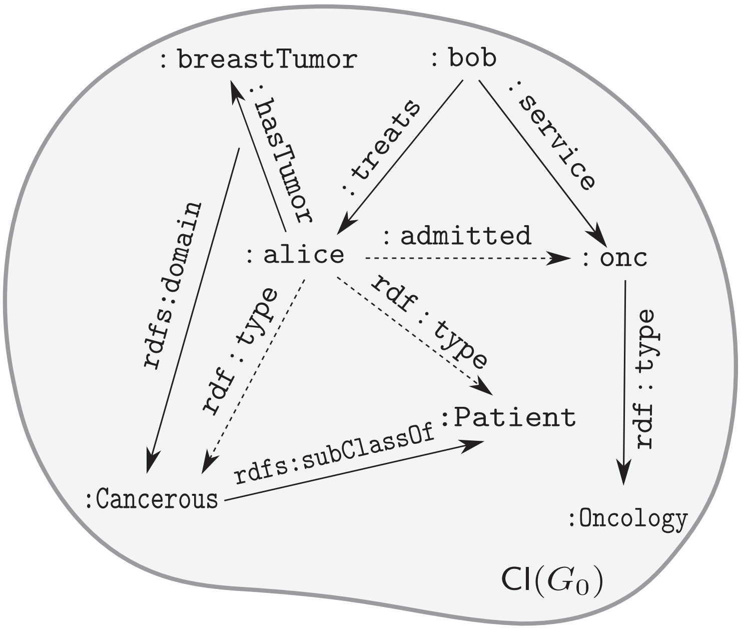
$$\frac{(\texttt{?d} \,;\texttt{:}\,\texttt{service}\,;\texttt{?s})(\texttt{?d}\,;\texttt{:}\,\texttt{treats}\,;\texttt{?p})}{(\texttt{?p}\,;\texttt{:}\,\texttt{admitted}\,;\texttt{?s})} = \mathbf{RAdm}$$

The rule is applied on G_0 of Figure 2.2 by replacing the variables with the terms of the graph.

$$\frac{(\texttt{:}\,\texttt{bob}\,;\texttt{:}\,\texttt{service}\,;\texttt{:}\,\texttt{onc})(\texttt{bob}\,;\texttt{:}\,\texttt{treats}\,;\texttt{alice})}{(\texttt{alice}\,;\texttt{:}\,\texttt{admitted}\,;\texttt{onc})}$$

Example 2.1.16 shows the closure computation of a graph using inference rules.

Example 2.1.16 *Consider the graph G_0 of Figure 2.2, and inference rules* **RDom** *and* $\mathbf{RSc_2}$ *in Table 2.1 (more details about these rules can be found in the next section). If we apply the inference rule* **RDom** *using triples et_1 and et_4 then we infer it_1. Afterwards,* $\mathbf{RSc_2}$ *is applied to et_2 and it_1 to infer it_3. Thus, $\mathsf{Cl}_{\{\mathbf{RDom},\mathbf{RSc_2}\}}(G_0) = G_0 \cup \{it_1, it_3\}$. Assume that we add rule* **RAdm**. *Referring to the graph G_0 of Figure 2.2, its closure now contains a new inferred triple $\mathsf{Cl}_{\{\mathbf{RDom},\mathbf{RSc_2},\mathbf{RAdm}\}}(G_0) = G_0 \cup \{it_1, it_3, it_2\}$. The new inferred triples are depicted by dashed arrows in Figure 2.4.*



	Subject	Predicate	Object
et_1	:hasTumor	rdfs :domain	:Cancerous
et_2	:Cancerous	rdfs :subClassOf	:Patient
et_3	:onc	rdf :type	:Oncology
et_4	:alice	:hasTumor	:breastTumor
et_5	:bob	:service	:onc
et_6	:bob	:treats	:alice
it_1	:alice	rdf :type	:Cancerous
it_2	:alice	:admitted	:onc
it_3	:alice	rdf :type	:Patient

Figure 2.4: An example of G_0 closure ($\mathsf{Cl}(G_0)$)

2.1.1.4 RDF Schema (RDFS)

RDF provides a way to express simple statements about resources, using named properties and values. However, RDF user communities also need a way to define the vocabularies (terms) they intend to use in those statements, specifically to indicate that they are describing specific kinds or classes of resources, and will use specific properties in describing those resources. RDFS is the schema language for RDF, and provides a way to specify the *vocabulary* (also known as *ontology*) that will be used in an RDF graph. It allows to define how individuals are related to one another, the properties we use to define our individuals and how they are related to other sets of individuals and to one another. In RDFS, a class corresponds to a group of resources

Table 2.1: Example of RDFS inference rules

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

$$\frac{(?\mathtt{p}\,;\mathtt{rdfs:range}\,;?\mathtt{r})\ (?\mathtt{x}\,;?\mathtt{p}\,;?\mathtt{y})}{(?\mathtt{y}\,;\mathtt{rdf:type}\,;?\mathtt{r})} = \mathbf{RRan}$$

$$\frac{(?\mathtt{c}_1\,;\mathtt{rdfs:subClassOf}\,;?\mathtt{c}_2)\ (?\mathtt{c}_2\,;\mathtt{rdfs:subClassOf}\,;?\mathtt{c}_3)}{(?\mathtt{c}_1\,;\mathtt{rdfs:subClassOf}\,;?\mathtt{c}_3)} = \mathbf{RSc}_1$$

$$\frac{(?\mathtt{a}\,;\mathtt{rdf:subClassOf}\,;?\mathtt{b})\ (?\mathtt{x}\,;\mathtt{rdf:type}\,;?\mathtt{a})}{(?\mathtt{x}\,;\mathtt{rdf:type}\,;?\mathtt{b})} = \mathbf{RSc}_2$$

$$\frac{(?\mathtt{p}_1\,;\mathtt{rdfs:subPropertyOf}\,;?\mathtt{p}_2)\ (?\mathtt{p}_2\,;\mathtt{rdfs:subPropertyOf}\,;?\mathtt{p}_3)}{(?\mathtt{p}_1\,;\mathtt{rdfs:subPropertyOf}\,;?\mathtt{p}_3)} = \mathbf{RSp}_1$$

$$\frac{(?\mathtt{p}\,;\mathtt{rdfs:subPropertyOf}\,;?\mathtt{q})\ (?\mathtt{x}\,;?\mathtt{p}\,;?\mathtt{y})}{(?\mathtt{x}\,;?\mathtt{q}\,;?\mathtt{y})} = \mathbf{RSp}_2$$

and is itself a resource (therefore identified by an IRI) belonging to the class `rdfs:Class`. An *instance* of a class is a resource belonging to that class. The `rdf:type` property states the relationship between instances and classes. Literals are defined by `rdfs:Literal`. For any given RDF triple, its subject and object are instances of `rdfs:Resource`, while its predicate is an instance of `rdf:Property`. The RDFS properties `rdfs:range` and `rdfs:domain` allow us to state the subject and the object class of a given `rdf:Property` respectively.

Example 2.1.17 *For instance* (`:hasTumor`;`rdfs:domain`;`:Patient`) *states that the domain of the property* `:hasTumor` *is the class* `:Patient`*, in other words the subject of any triple with predicate* `:hasTumor` *is an instance of* `:Patient`*.*

RDFS allows us to define classes and properties hierarchies by the properties `rdfs:subClassOf` and `rdfs:subPropertyOf` respectively. `rdfs:subClassOf` allows to say that the class extension of a class description is a subset of the class extension of another class description. For instance (`:Patient`;`rdfs:subClassOf`;`:Person`) means that any instance of `:Patient` is an instance of `:Person`. Similarly, `rdfs:subPropertyOf` states that the property extension of a given property is a subset of another property extension. For instance (`:hasTumor`;`rdfs:subPropertyOf`;`:hasDisease`) means that any two resources related with `:hasTumor` are related with `:hasDisease`. RDFS allows reasoning over RDF triples using a set of defined inference rules (aka *entailment rules*). Table 2.1 shows a subset of RDFS inference rules used in our examples. We refer the reader to [Hayes 2014] for the rest of the rules.

RDom states that if there exists a triple with a given property, then its subject is an instance of this property domain. **RRan** is similar to **RDom** and generates the fact that the triple object is an instance of the property range. $\mathbf{RSc_1}$ and $\mathbf{RSp_1}$ define the subclass and subproperty transitivity respectively. $\mathbf{RSc_2}$ defines the type propagation, in other words if a resource is an instance of a given class, then it is also an instance of the parents of this class. $\mathbf{RSp_2}$ defines the property propagation, which mean that if two resources are related with a property then they are related with the parents of this property.

2.2 Access control

Access control refers to the process of regulating access to protected data based on pre-defined security policies. The advantage of using policies is that the system behavior can be managed without the need of reimplementation. The development of an access control system requires the definition of the regulations according to which access is to be controlled and their implementation as functions executable by a computer system. The development process is usually carried out with a multi-phase approach based on the following concepts [Samarati 2001]:

- Security policy: it describes the most abstract view of the system. At this level access control rules are defined. The requirements of the system are described in order to comply with some specification (e.g., laws, regulations). This description does not provide any method on how it should be enforced.
- Security model: it formalizes the rules defined in the security policy and describes the way they should work. This level aims at providing a framework where proof of properties could be accomplished.
- Security mechanism: it describes the low level methods that are used to enforce the rules formalized at the security model level.

2.2.1 Access control models

The most well-known access control models in the literature are : Identity Based Access Control (IBAC) where the access to objects is based on the identity of the user, Mandatory Access Control (MAC) where access is based on security levels assigned to both users and resources and Role Based Access Control (RBAC) where authorizations are granted to roles instead of users. Other more flexible models have been introduced such as Attribute Based Access Control [Wang 2004] and Organization Based Access Control (OrBAC) [Cuppens 2003].

In the following we give more details on IBAC, MAC, RBAC and ABAC models.

2.2.1.1 Identity-Based Access Control (IBAC)

In the IBAC model, the access to objects is based solely on the identity of the subject and the rights specified for that identity on each object. In this model, privileges can be passed from a subject to another, where an administrative policy regulates grants and revocations of the privileges. The access control matrix provides a basic framework for describing IBAC. It was first proposed by Lampson [Lampson 1974] for resource protection within operation systems. In this model, authorizations are represented as a matrix $|S| \times |O|$ where $|S|$ is the set of subjects and $|O|$ the set of objects. The matrix is arranged as a two-dimensional array where each row is labeled by a subject and each column labeled by an object. Each entry of the matrix specifies the actions of the subject on the object. For instance, a matrix entry for s and o which contains `read`, grants to s the right to read o. The drawback of the access control matrix is that it will be enormous in size if used in large systems and most of its cells are likely to be empty. In practical systems, there exists multiple ways to implement the access control matrix. A popular approach is using ACLs (Access Control List). An ACL is associated with an object and consists of a number of entries defining the rights assigned to each subject on that object. Another way to implement the access control matrix is the C-List (Capability list) where each subject is associated with a list of objects and the rights that the subject has on them.

2.2.1.2 Mandatory Access Control (MAC)

A different approach of controlling access to resources is the MAC model in which access to resources is controlled based on the perspective attributes of the subject and object. Such approach was motivated by the problem of Trojan Horses which the DAC model suffers of. Unlike DAC where the owner defines the access rights, in MAC accesses are centrally controlled. In this model, mandatory policies govern access on the basis of subjects and objects classifications. Each user and each object is assigned a security level. The security level associated with the object reflects the sensitivity of the information it contains. The security level associated with a user reflects the user's clearance. The MAC model is usually associated with the Bell-LaPadula Model (BLP) [Bell 1973]. In the BLP model clearance and sensitivity levels take values from the set of *access classes*.

Definition 2.2.1 (Access class) *An access class consists of two components : a security level and a category set. The security level is an element of a totally ordered set of levels. The category set is a subset of an unordered set specific to the application area.*

The set of the access classes is partially ordered according to relation called *dominance relation*.

Definition 2.2.2 (Dominance relation) *Let L and C be the set of security levels and categories respectively. Let $c_i = (L_i, SC_i)$ and $c_k = (L_k, SC_k)$, with $L_i, L_k \in L$ and $SC_i, SC_k \subseteq C$, be two access classes. We say that c_i* dominates *c_k denoted by $c_i \geq c_k$ if the following holds :*

- *$L_i \geq L_k$ the security level of c_i is greater than or equal to the security level of c_k;*
- *$SC_i \supseteq SC_k$ the category set of c_i includes the category set of c_k.*

Access classes with the dominance relationship between them therefore form a lattice [Denning 1976].

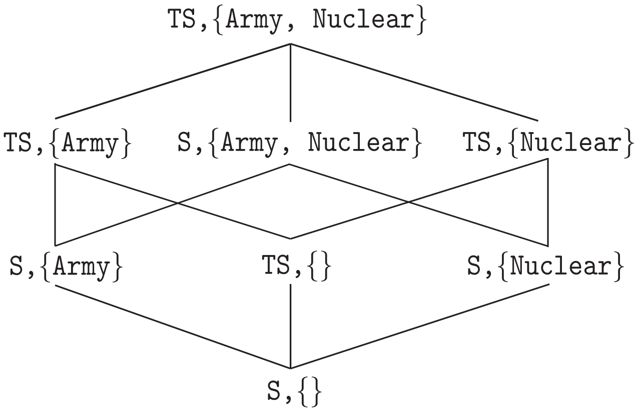


Figure 2.5: Security lattice example

Example 2.2.3 *Figure 2.5 shows an example of a security lattice. The seurity levels are defined as follows {`Top Secret (TS)`, `Secret (S)`, `Confidential (C)`, `Unclassified (U)`} where `TS > S > C > U`. The category set refers to the military application domain : {`Army`, `Nuclear`}.*

In the context of flow control, the BLP model follows two principles that are required to hold.

- no read-up : The subject clearance must dominate the object sensitivity level, in other words a subject s with an access class c_s can read an object with an access class c_o if $c_s \geq c_o$.
- no write-down : The subject clearance must be dominated by the object sensitivity level, *i.e.* a subject s with an access class c_s can write an object with an access class c_o if $c_o \geq c_s$.

Satisfaction of these principles prevents information flows from high levels to low levels.

2.2.1.3 Role-Based Access Control (RBAC)

The principle purpose of RBAC is to specify and enforce enterprise-specific security policies in a way that maps naturally to an organization's structure. Moreover, RBAC facilitates security administration by granting authorizations to roles instead of individual users. Role based policies regulate users' access to the information on the basis of the activities the users execute in the system. A role can be defined as a set of actions and responsibilities associated with a particular work activity. Access authorizations on objects are specified for roles and a user playing a role is allowed to execute all accesses assigned to this role. A user can have multiple roles and similarly a role can be granted to different users. Some RBAC approaches allow users to exercise

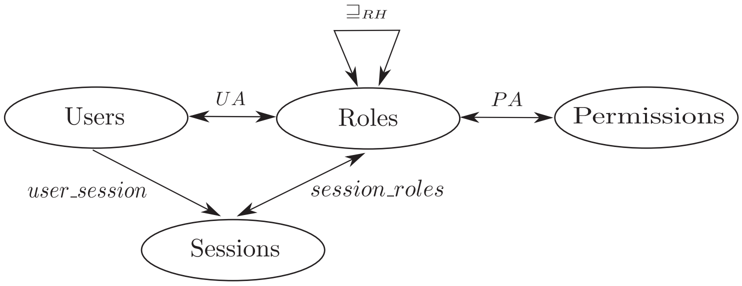


Figure 2.6: RBAC architecture

multiple roles at the same time, in addition other approaches limit the user to use one role only at a time. The concept of grouping privileges was initially proposed by [Baldwin 1990], and different types of RBAC models have been proposed in the literature [Ferraiolo 2001], [Sandhu 1996].

Figure 2.6 illustrates the basic concepts of standard RBAC which is defined by a set of users U, a set of roles R and a set of permissions P. A permission is an approval of a particular mode of access to one or more objects in the system. Users are associated with roles using relation UA. Similarly, permissions are associated with roles by relation PA. Users activate sessions to interact with RBAC system. A session is a mapping of one user to possibly many activated roles. The activated roles determine which permissions are available to the user at a given time during the session. Administration has been further reduced by the use of roles hierarchy to allow the propagation of access control privileges. Role hierarchies are a natural means for structuring roles to reflect an organization's lines of authority and responsibility [Sandhu 2000]. The hierarchy is defined as a partial order relation $\sqsupseteq_{RH}$ on R. A role inherits all permissions of less-powerful (junior) roles which avoids the need to assign these permissions explicitly. This reduces the administration overhead, but on the other hand increases the enforcement overhead because the permissions of the junior roles need to be considered in the decision computation. In this case a user is authorized for permission p if there exist roles r and r' such that $(u, r) \in UA$, $r \sqsupseteq_{RH} r'$ and $(p, r') \in PA$.

2.2.1.4 Attribute Based Access Control (ABAC)

Whereas traditional access control systems were based on the identity of the requester or through predefined attribute types such as roles or groups assigned to that requester, in open environments such as the Internet, this approach is not effective because often the requester and the resource belong to

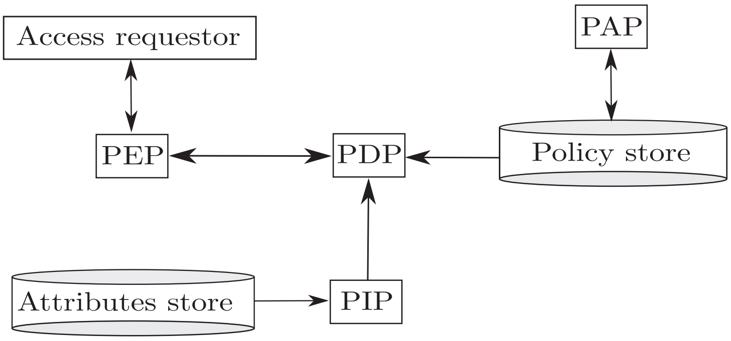


Figure 2.7: XACML Architecture

different domains.

It has also been noted that the requester qualifiers of identity, groups, and roles are often insufficient in the expression of real-world access control policies. An alternative is to grant or deny user requests based on arbitrary attributes of the user and arbitrary attributes of the object, and environment conditions that may be globally recognized and more relanevant to the policies at hand. This approach is often referred to as ABAC. The main advantage of ABAC is enabling object owners or administrators to apply access control policy without prior knowledge of the specific subject and for an unlimited number of subjects that might require access. One example of an access control framework that is consistent with ABAC is the Extensible Access Control Markup Language (XACML) [Ramli 2011]. XACML architecture consists of several logical components (see Figure 2.7).

First of all, a reference monitor concept is used to intercept access requests. This component is called a Policy Enforcement Point (PEP). Access requests are transmitted from the PEP to a Policy Decision Point (PDP) for retrieval and evaluation of applicable policies. Policies are specified and stored in Policy Administration Point (PAP). The PDP gets the attributes of subjects, objects, and the environment from the Policy Information Points (PIP). The XACML policy language is based on three main elements: PolicySet, Policy, and Rule. A PolicySet is a set of single policies or another PolicySet. Policies are sets

```
<Request>
  <Subject>
    <Attribute AttributeId=":1.0:subject:subject-id">
      alice
    </Attribute>

    <Attribute AttributeId=":subject:role" DataType="http://www.w3.org/2001/
        XMLSchema#string">
      <AttributeValue>
        nurse
      </AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId=":resource:resource-id"
    DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        record1
      </AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <AnyAction />
  </Action>
</Request>
```

Figure 2.8: XACML Request example

```
<Policy PolicyId="policy_patients" RuleCombiningAlgId=":rule-combining-
    algorithm:deny-overrides">
  <Description>Policy for the cancerous patients </Description>
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId=":function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            record1
          </AttributeValue>
          <ResourceAttributeDesignator>
            :resource:resource-id
          </ResourceAttributeDesignator>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule RuleId="permit_to_nurses" Effect="Permit">
    <Description>Permit access from nurses</Description>
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId=":function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
                ">
              nurse
            </AttributeValue>
            <SubjectAttributeDesignator
                        DataType="http://www.w3.org/2001/XMLSchema#string">
              :subject:role
            </SubjectAttributeDesignator>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
  </Rule>
</Policy>
```

Figure 2.9: XACML policy example

of single Rules which have a Condition, an Effect, and a Target. To find the relevant policy for an access control request, every PolicySet, Policy, and rule has a Target, which is evaluated at the access request time. A Target consists in a specification of sets of subjects, objects, operations, and environment using their respective attributes which can be evaluated with match functions. Conditions can be used beyond the Target to further specify the applicability of a Rule using predicates, while Effects denote the result of a Rule, e.g. permit or deny.

Example 2.2.4 *Figure 2.8 shows an example of a XACML request, from the subject alice who is a nurse, requesting access to record1. Figure 2.9 is an example of a policy targeting record1. The policy contains one rule which permits access to nurses. Hence alice request will return permit.*

When the relevant Policies and Rules are found, they are evaluated independently of each other; contradicting evaluation results can be resolved using policy combining algorithms which apply *conflict resolution strategies*. For instance, the policy of Figure 2.9 resolves conflicts with deny-overrides algorithm. In the next section we describe the conflict resolution and we give details about the strategies known in the literature.

2.2.2 Conflict resolution

Negative authorizations have been introduced by [Bertino 1997] to extend the System R access control model by the possibility of specifying explicit denials. This feature enables adding of exceptions in existing permissions. An access control model that supports positive and negative authorizations has a *sign* (*effect*) field in permission tuple.

Definition 2.2.5 (Authorization tuple) *From a conceptual point of view, an authorization is defined by the tuple* $\langle$*subject; sign; access right; object*$\rangle$*. The sign is either positive or negative, and determines whether the subject can perform the access right on the object.*

Traditionally, positive and negative authorizations have been used in mutual exclusion using a *default policy* that determines the sign of a permission corresponding to two classical approaches namely: [Samarati 2001]:

- *Closed Policy*: denies all accesses, unless a corresponding positive authorization permits it.
- *Open Policy*: a policy where accesses are by default allowed, and denied if there exists an explicit negative authorization.

The open policy has usually found application in those scenarios where the need for protection is not strong and by default access is to be granted. The closed policy is the mostly adopted where denying access by default ensures better protection. In the recent access control models, negative and positive authorizations are combined to handle exceptions, giving rise to other issues, namely:

- *Completeness* : how to treat objects which have no defined authorization?

- *Consistency* : how to treat objects which are both allowed and denied at the same time?

In order to achieve completeness, one has only to define the above-cited default policy. Consistency is more complex and it is achieved by defining a *Conflict Resolution Strategy.* Conflict resolution consists in choosing one of the conflicting authorizations to make a final access decision. Below are examples of known conflict resolution strategies in the access control literature.

- *Denials Take Precedence* (`DTP`): in this case, negative authorizations are always adopted when a conflict occurs. In other words the principle says that if we have one reason to authorize an access and another to deny it then we deny it.
- *Permissions Take Precedence* (`PTP`): in this case, positive authorizations are always adopted when a conflict occurs. In other words the principle says that if we have one reason to authorize an access and another to deny it then we authorize it.
- *First Applicable* (`FA`):in this case, the first applicable authorization in order of presentation is picked.
- *Most Specific Takes Precedence* (`MSTP`): in this case it is supposed that subjects or objects are hierarchically related. The *most specific* authorization w.r.t. the hierarchical relation should be the one that prevails.

A strategy may not achieve consistency such as with `MSTP` where a partial order is supposed to exist between authorizations. The conflict may still remain if the conflicting authorizations are incomparable. Since conflict resolution strategies are not mutually exclusive, one can decide to combine strategies together to achieve a more effective conflict resolution. Thus a strategy chain can be constructed to solve the issue. For instance, one can decide to try solving conflicts with the `MSTP` first, and apply the `DTP` principle on the remaining conflicts (*i.e.* conflicting authorizations that are not hierarchically related).

2.3 Access control for RDF data

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 [Jain 2006]. 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 [Polleres 2007], and then rely on access control mechanisms for deductive databases, such as the one by Barker [Barker 2002]. Unfortunately, it seems that problems that arise when dealing with RDF data have 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 have been proposed:

- Abel *et al.* [Abel 2007] propose a query rewriting mechanism to enforce authorizations. Their framework evaluates the applicable policies and expands the query depending on the result of the evaluation. The modified query is then sent to the RDF store which executes it like a usual RDF query. The authors used a specific language to define their policies.

- Flouris *et al.* [Flouris 2010] propose an annotation based access control language with its formal semantics for fine-grained authorizations on RDF data. In their proposal, authors propose to enforce their policy. The triples are specifically annotated as accessible or not accessible using access control permissions.

- Costabello *et al.* [Costabello 2012] propose a context-aware access control model. The authors present an ontology based on existing vocabularies and relies on SPARQL `ASK` queries to determine whether the requester has the necessary attributes to access the resource. They use context information to rewrite the user SPARQL query which is executed over the accessible named graphs only.

- Reddivari *et al.* [Reddivari 2005] propose an access control language for RDF stores that considers update operations. They use meta-rules to define conflict resolution strategies and default policies. They propose a query-time approach, where each triple in the user query result is checked whether it is accessible or not.

- Lopes *et al.* [Lopes 2012] propose an annotation approach using an extended version of RDF called *Annotated RDF* [Udrea 2010]. They propose an access control annotation domain where each triple is annotated with a label and labels are propagated through inference rules.

The annotated triples are queried using an extended SPARQL language AnQL [Lopes 2010].

- Papakonstantinou *et al.* [Papakonstantinou 2012] propose a flexible model that defines the access label of a triple as an algebraic expression. Their model assigns abstract labels to RDF triples and computes the access decision using abstract operators that encode inference and propagation. Their main contribution is the efficient handling of updates by easy determination of the labels that are affected by these updates. When a triple is assigned different tokens, they use a conflict resolution operator which returns one concrete token that represents the final decision.

- Jain *et al.* [Jain 2006] propose a label-based model to control access to RDF data. Security labels are assigned to graph patterns. The patterns are mapped to the triples to determine their security classifications. They propose an algorithm that detects unauthorized inferences where higher security triples may be inferred from lower security triples.

2.3.1 Comparison of related works

In order to study the works related to the domain of controlling access to RDF data, we defined a set of comparison criteria. The study summary is shown in Table 2.2. In the following, we give the details about the criteria and the results of the study.

Authorization object An important aspect of access control models is their granularity with respect to the protected objects. Most of the works define authorizations over graph patterns to ease administration. [Jain 2006] and [Reddivari 2005] define their authorizations over simple Triple Patterns which does not allow the specification of expressive policies. For instance an authorization such as *Deny access to patients records if they have cancer* cannot be specified. [Flouris 2010], [Abel 2007] and [Papakonstantinou 2012] models are more expressive by using BGPs in their authorizations. [Costabello 2012] authorizations are coarse-grained as they are defined over Named Graphs.

Triples protection The purpose of this part of study is to check whether the proposed models consider the implicit triples or only explicit ones. As the semantics of an RDF graph are given by its closure, it is important for an access control model to take into account the implicit knowledge held by

Table 2.2: Comparison of the approaches

Criteria		Approaches						
		Farkas	Reddivari	Flouris	Abel	Costabello	Lopes	Papakonstantinou
Authorization object	Triple Pattern	•	•	•	•		-	•
	BGP			•	•		-	•
	Named Graph					•	-	
Triples protection	Explicit	•	•	•	•	•	•	•
	Implicit	•	•				•	•
Considered rules	RDFS	•	•				•	•
	User defined						•	
Inference leakage detection		•						
Default policy	Open		•	•		-	-	•
	Closed	•	•	•	•	-	-	•
Conflict resolution	PTP		•	•		•		•
	DTP	•	•	•	•		•	•
Language semantics	Syntax	•	•	•	•	•		•
	Semantics	•		•		•	•	•
Subject specification	Not specified	•		•				•
	Attributes		•		•	•	•	
Actions	Read	•		•	•		•	•
	CRUD		•			•		
Query language	SPARQL	•		•	•	•	•	•
	SeRQL	•		•	•			
	RDQL	•	•					
Enforcement approach	pre-processing				•	•		
	post-processing		•					
	annotation-based	•		•			•	•

this graph. In the Semantic Web context, the policy authorizations deny or allow access to triples whether they are implicit or not. In [Reddivari 2005] model, the implicit triples are checked at query time. Inference is computed during every query evaluation, and if one of the triples in the query result could be inferred from a denied triple, then it is not added to the result. In the [Jain 2006] label-based model the implicit triples are automatically labeled on the basis of the labels assigned to the triples used for inference. [Lopes 2012] propose an approach inspired from provenance where each triple is annotated with a label and labels are propagated through inference rules to the implicit triples. [Papakonstantinou 2012] propose a flexible model that defines the access label of a triple as an algebraic expression (abstract tokens). The labels are propagated to the implicit triples through the inference rules. [Flouris 2010], [Abel 2007] and [Costabello 2012] consider explicit triples only.

Considered inference rules In this part of study, we examined the proposed models with respect to the supported inference rules. All the analyzed works that consider inference, support the RDFS rules only except [Lopes 2012], who extend inference rules with labels in order to propagate them to implicit triples. Their model allows to specify a custom rules in order to provide application specific inferencing.

Inference leakage detection As specified earlier, the inference leakage problem arises when denied triples are inferred from the accessible ones. [Jain 2006] was the only work that considered the inference leakage problem. They proposed an algorithm that detects unauthorized inferences by checking if triples with high security label, may be inferred from lower security triples.

Default policy As described in Section 2.2.2, a Default Policy is used to achieve completeness of the access control model. The Default Policy in the models proposed by [Jain 2006] and [Abel 2007] is hard-coded, whereas in the [Reddivari 2005], [Flouris 2010] and [Papakonstantinou 2012] models, it can be specified by administrators. [Costabello 2012] and [Lopes 2012] do not mention how they treat the triples without defined authorizations.

Conflict resolution Conflict resolution strategies allow us to remove inconsistencies that may occur when multiple authorizations with different effects are applicable to the same triples. In the model of [Jain 2006] a partial order is defined between security labels, and when more than one pattern maps to the same triple, the most restrictive or the lowest upper bound takes precedence. Unfortunately, their conflict resolution is hard-coded. [Costabello 2012] and [Lopes 2012] resolve conflicts by hard-coding the DTP strategy, whereas [Reddivari 2005] and [Flouris 2010] models allow the specification of PTP or DTP strategies only. [Papakonstantinou 2012] define the strategies at an abstract level which is mapped to concrete strategies such as DTP or PTP.

Language semantics Access control models use policy languages to define which objects are accessible and which are not. Only [Jain 2006], [Costabello 2012] and [Flouris 2010] gave the formal semantics of their language.

Subject specification Information about the subject is used by the access control system to determine its accessible objects. It can be based on simple credentials such as user name and password, or on context information and attributes. [Costabello 2012], [Abel 2007], [Reddivari 2005] and [Lopes 2012] use attribute-based approach to determine the accessible triples of the requester, whereas [Papakonstantinou 2012] and [Jain 2006], [Flouris 2010] did not consider subject specification.

Actions The purpose of access control mechanisms is to check whether or not the subject has the authorization to perform the action on the data. The actions that can be performed on RDF data are `Create`, `Read`, `Update` and `Delete` (CRUD). Since version 1.1, updates are supported in SPARQL by the use of keywords : `insert`, `modify` and `delete`. The model proposed by [Reddivari 2005] supports RDF updates. They consider insertion by defining the three actions, namely, insert for adding a triple, insertModel for adding an implicit triple and insertSet for adding a set of triples. Regarding triples deletion, they defined three actions, namely remove for removing a triple, removeModel for removing an implicit triple and removeSet for removing a set of triples. [Costabello 2012] also consider RDF updates using SPARQL 1.1 update language.

Query language In this part of study we examined the different works with respect to the supported query languages. [Lopes 2012] demonstrate how AnQL [Lopes 2010] an extension of the SPARQL query language can be used to enforce access control, by rewriting using the requesters credentials to rewrite a SPARQL query to an AnQL query. [Costabello 2012] express access conditions as SPARQL `ASK` queries in order to determine the authorized named graphs of the user querying the SPARQL endpoint. Besides SPARQL, [Flouris 2010] and [Abel 2007] support SeRQL query language whereas [Reddivari 2005] supports RDQL only.

Enforcement approach In this part of study, we examined the different works with respect to the used enforcement approach. The latter can be *pre-processing*, *post-processing*, or *annotation based*.

- The pre-processing approaches enforce the policy before evaluating the query. For instance, the query rewriting technique consists in reformulating the user query using the access control policy. The new reformulated query is then evaluated over the original data source returning the accessible data only. This technique was used by [Costabello 2012] and [Abel 2007] where the user query is rewritten with respect to the policy, and then evaluating the expanded query on the original dataset.

- In the post-processing approaches, the query is evaluated over the original data source. The result of the query is then filtered using the access control policy to return the accessible data. [Reddivari 2005] use a post-processing approach by evaluating the query over the original graph, filtering the triples in the query result and then returning the authorized triples to the requester.

- The rest of the works use annotation approach to enforce their models. In this case, every triple is annotated with access control information. During query evaluation, only the triples annotated with a permit access are returned to the user. In the label-based model proposed by [Jain 2006], each triple is annotated with a label that represents its security classification. [Flouris 2010] annotate each triple with a boolean stating whether it is accessible or not. [Lopes 2012] annotate the triples with non-recursive Datalog with negation programs which evaluation decision defines whether the triple is accessible or not. [Papakonstantinou 2012] annotate the triples with *abstract labels* that encode inference and propagation of labels along the RDFS inference rules. To evaluate the label, each application provides its own *concrete policy* and semantics which allows the application to decide whether a triple is accessible or not.

2.4 Study summary

Regarding the protection of implicit triples, we mentioned that the propagation techniques assign automatically an accessibility label to the implicit triples on the basis of the triples used to infer them. Hence if one of the triples used for inference is denied, then the inferred triple is also denied even if it is explicitly authorized. This goes against the intuition that implicit triples are part of the knowledge held by the graph, hence they must be regarded as the explicit triples and not depend on the triples used to infer them. We illustrate with the following example.

Example 2.4.1 *Let us consider the graph G_0 of Figure 2.2. Suppose we want to protect G_0 by applying the policy P =*{deny access to triples having `:breastTumor` property, allow access to all resources which are instance of `:Patient`}*. If we apply the inference rules* **RDom** *and* **RSc₂** *we get $\mathsf{Cl}_{\{\mathbf{RDom},\mathbf{RSc_2}\}}(G_0) = G_0 \cup \{it_1, it_3\}$. With the propagation approaches which consider inference [Lopes 2012, Papakonstantinou 2012], the triple it_3 =*(`:alice` ; `rdf :type` ; `:Patient`)} *will be denied since it is inferred from denied triples (et_4). Hence the fact that alice is a patient will not be returned in the result even though the policy clearly allows access to it.*

Another problem which the propagation techniques suffer from, is the inference leakage. In fact, even some implicit triples are labeled as denied by the policy, they could be inferred from triples labeled as accessible. To illustrate, consider the following example.

Example 2.4.2 *We want to protect the graph G_0 of Figure 2.2 in presence of* **RDom** *inference rule, using the policy P =*{allow access to triples having `:breastTumor` property, deny access to all resources which are instance of `:Cancerous`}*. As the closure of the graph $\mathsf{Cl}_{\{\mathbf{RDom}\}}(G_0) = G_0 \cup \{it_1\}$, there is one implicit triple i.e. it_1. With the propagation techniques which consider inference [Lopes 2012, Papakonstantinou 2012, Jain 2006], it_1 will be assigned two labels, a deny label from the policy and a permit label inherited from the premises et_1 and et_4. After resolving conflicts, it_1 will be denied and will not be returned to the requester, whereas et_1 and et_4 will be. Using a local reasoner, the requester could apply* **RDom** *on et_1 and et_4 to infer it_1, hence an information leakage through inference. Note that even [Reddivari 2005] query-time approach suffer from the inference leakage problem.*

The propagation approaches that consider the inference leakage problem such as [Jain 2006], propose solutions after the labeling operation. In fact, they check the labeled graph to detect information leakage.

Regarding the conflict resolution, it is clear that different approaches can be taken to deal with positive and negative authorizations. Hard-coding the conflict resolution strategy makes the access control model less expressive. Moreover, even giving the choice to the administrator over pre-defined strategies would not make the model more expressive. Indeed, there are always situations where the pre-defined strategies do not fit.

As regards the supported query languages, the approaches that are based on query rewriting techniques support one or a limited number of languages.

Concerning policy enforcement, the pre-processing approaches such as query rewriting, are tied to the used query language. Changing the query language would lead to an update of the policy enforcer. Furthermore, the execution of a the reformulated query can be computationally expensive depending on the number of triple patterns in the query and size of the triple store. For instance, [Abel 2007], rewrite SeRQL queries using path expressions found in the body of the user's applicable policies, combined with "OR" keyword. A query with a triple pattern ?s ?p ?o will be reformulated to a query that may be computationally expensive since it combines all the possible policies which leads to a long path expression. Moreover, not all the user queries can be handled. More complex queries such as path queries are not supported.
In the post-processing approaches, policies conditions are usually not based on the data content but on the subject attributes [Reddivari 2005]. The query is evaluated on the RDF store, and the policies are checked afterward on the result triples. The query answer time may be considerably too large. As an example, suppose an unauthorized user submits a query asking for all available triples in the store. A post-processing approach would retrieve all the triples first and then filter them all out.

The data annotation approaches that use custom language such as [Lopes 2012], rewrite the user query to include annotations based on the access control policy. The query that contains annotations is then evaluated over an extension of RDF that supports annotations. Similarly to the rewriting technique, these approaches are tied to the query language. Moreover, additional mechanisms are needed in the RDF store to support custom languages and extended RDF models. Other approaches use the graph name position to store a boolean representing the access decision [Flouris 2010]. Which means that for each triple, all the user profiles decisions must be computed and stored at the design time. Moreover, this kind of approaches does not support incremental re-computation of annotations, as the latter do not store any information about the policy.

2.5 Filling the gaps

Our formal approach and its concrete enforcement are driven by the following key requirements: *expressiveness*, *modularity*, *applicability*, *verifiability*, and *performance*.

- *Expressiveness*: We propose in Chapter 3 the syntax and semantics of an expressive fine-grained access control model for RDF. In our model, authorizations are defined using SPARQL BGPs, which allows the definition of *fine-grained policies*. Moreover, instead of hardcoding the conflict resolution strategy or selecting one from predefined strategies, we proposed a more liberal approach. Indeed, abstracting the conflict resolution function allows the administrator to define not only the classical conflict resolution strategies but also custom strategies, which makes our model more expressive.

- *Modularity*: Our model semantics are defined by means of positive subgraph which relies on the access control policy and the base graph, without reference to a concrete query language such as SPARQL, in contrast to models driven by query rewriting. In our model, the supported set of inference rules is not be limited to RDFS rules. User defined rules can be used as well. Indeed, to cope with inference, it suffices to replace the base graph G by its closure $\mathsf{Cl}(G)$ according to a set of inference rules. This makes our model independent from the entailment engine as well. Moreover the implicit triples are considered as the explicit ones and do not depend on the triples used to infer them. In our model, the entity to which the authorizations are granted or denied is left implicit. The upstream mapping from requesters to authorizations may use any model from the literature. In Chapter 5 we propose a XACML-inspired policy language that allows the definition of subject attribute-based policies.

- *Verifiability*: In Chapter 4, we formally characterize the issue that arises when inference rules produce facts which would have been forbidden otherwise. This issue occurs when the positive subset of a closed graph is not, itself, closed. We show that it can be statically checked, without knowledge of the base graph G, whether a policy is consistent w.r.t. a set of inference rules.

- *Applicability*: Our model is defined with triple-based authorizations which are both natural for SPARQL knowledgeable administrators and are naturally converted to efficient SPARQL `CONSTRUCT` queries to be run on the store. In Chapter 3, we show that our policies can capture quite complex access control requirements with exceptions that occur in real-life scenarios. We propose in Chapter 6 a data-annotation-based enforcement approach to our model and we show that no additional mechanisms are needed to apply our enforcement.

- *Performance*: We focus our attention on search queries on graphs. We show in Chapter 6 that our implementation incurs *reasonable overhead* at runtime (about +50%) with respect to the optimal solution which consists in materializing the user's accessible subgraph. We show that the query evaluation overhead is independent from the size of the base graph and the number of policy authorizations.

Chapter 3

A fine-grained access control model for RDF stores

Contents

▷ *In this chapter, we define an access control model for RDF called* ***`AC4RDF`*** *(Access Control For the Resource Description Framework), that uses the ingredients from Section 2.1 (P. 14). First, we define atomic authorizations and policies, then we give their formal semantics. Conflict resolution strategies are used to resolve such conflicts by selecting the preferred authorizations to apply, with respect to some properties of the authorizations. Whereas most of the works hard-code their strategies, others define them as parameters that are fixed by the administrator during policy design. We propose a more liberal approach by defining our policy using an* abstract conflict resolution function ch *which is defined by the administrator. We present three conditions that must be respected by the authorization policy to be well-formed. In Section 3.2 we give examples of how to build* ch *to apply simple strategies such as* ***`DTP`****. Moreover, we show how to build more elaborate strategies such as* ***`MSTP`*** *to handle exceptions.* ◁

3.1 Authorization policy

An authorization policy is an encoding of the control requirements of a application using the authorization language that is understood by the PDP. An authorization is defined as a tuple made of an effect, some resource definition and a condition. The resources under consideration are triples from the RDF graph being accessed by the requester. These triples are selected by means of triple patterns. The conditions are graph patterns that tell when the authorization is applicable. The effect is a boolean value denoting giving or forbidding action to the selected resources.

In this chapter, we assume that the PDP knows what are the authorizations applicable to a given authenticated requester. The entity to which authorizations are granted or denied is left implicit. 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. Regarding upstream policy definitions, we propose in Chapter 5 an attribute based high level language that allows to define global policies. When a user requests access, her/his policy is enforced by `AC4RDF`.

3.1.1 Authorization semantics

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 without additional query mechanisms.

Definition 3.1.1 (Authorization) *Let* $\mathsf{Eff} = \{+,-\}$ *be the set of applicable* effects. *Formally, an* authorization $\mathscr{a} = (e, h, b)$ *is a element of* $\mathsf{Auth} = \mathsf{Eff} \times \mathsf{TP} \times \mathsf{BGP}$*. The component e is called* the effect *of the authorization* $\mathscr{a}$*, 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* $\mathsf{hb}(\mathscr{a}) = \{\mathsf{head}(\mathscr{a})\} \cup \mathsf{body}(\mathscr{a})$ *the* underlying graph pattern *of the authorization* $\mathscr{a}$*.*

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

Example 3.1.2 *Consider the set of authorizations shown in Table. 3.1. Authorization $\mathscr{a}_1$ grants access to triples with predicate* `:hasTumor`*. Authorization $\mathscr{a}_2$ states that all triples of type* `:Cancerous` *are denied. Authorizations $\mathscr{a}_3$ and $\mathscr{a}_4$ state that triples with predicate* `:service` *and* `:treats` *respectively are permitted. Authorization $\mathscr{a}_5$ states that triples about admission to the oncology service are specifically denied, whereas the authorization $\mathscr{a}_6$ states that such information are allowed in the general case. $\mathscr{a}_7$ grants access to properties domain and $\mathscr{a}_8$ denies access to any triple which object is* `:Cancerous`*. Finally, authorization $\mathscr{a}_9$ denies access to any triple, it is meant to be a default authorization.*

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

Definition 3.1.3 (Applicable Authorizations) *Given a finite set of authorizations $\mathscr{A} \in \mathscr{P}(\mathsf{Auth})$ and a graph $G \in \mathsf{BGP}$, the function* ar *assigns to each triple $t \in G$, the subset of* applicable authorizations *from $\mathscr{A}$:*

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

Table 3.1: Example of authorizations

$$
\begin{aligned}
\mathscr{a}_1 &= \texttt{GRANT}(\texttt{?p}\,;\texttt{:hasTumor}\,;\texttt{?t})\\
\mathscr{a}_2 &= \texttt{DENY }(\texttt{?p}\,;\texttt{rdf:type}\,;\texttt{:Cancerous})\\
\mathscr{a}_3 &= \texttt{GRANT}(\texttt{?d}\,;\texttt{:service}\,;\texttt{?s})\\
\mathscr{a}_4 &= \texttt{GRANT}(\texttt{?d}\,;\texttt{:treats}\,;\texttt{?p})\\
\mathscr{a}_5 &= \texttt{DENY }(\texttt{?p}\,;\texttt{:admitted}\,;\texttt{?s})\\
&\quad\ \texttt{WHERE}\,\{(\texttt{?s}\,;\texttt{rdf:type}\,;\texttt{:Oncology})\}\\
\mathscr{a}_6 &= \texttt{GRANT}(\texttt{?p}\,;\texttt{:admitted}\,;\texttt{?s})\\
\mathscr{a}_7 &= \texttt{GRANT}(\texttt{?p}\,;\texttt{rdfs:domain}\,;\texttt{?s})\\
\mathscr{a}_8 &= \texttt{DENY }(\texttt{?s}\,;\texttt{?p}\,;\texttt{:Cancerous})\\
\mathscr{a}_9 &= \texttt{DENY }(\texttt{?s}\,;\texttt{?p}\,;\texttt{?o})
\end{aligned}
$$

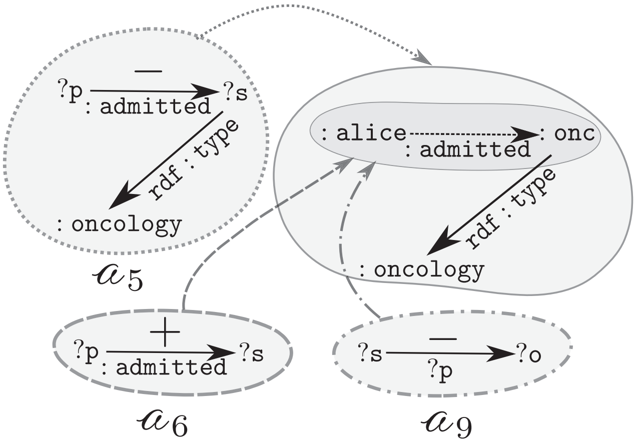


Figure 3.1: Authorizations applicable to it_2

Example 3.1.4 *Consider the graph* $\mathsf{Cl}(G_0)$ *shown in Figure 2.4 (P. 23) and the set of authorizations* $\mathscr{A}$ *shown in Table 3.1. The applicable authorizations on triple* it_2 *are computed as follows :* $\mathsf{ar}(\mathsf{Cl}(G_0), \mathscr{A})(it_2) = \{\mathcal{a}_5, \mathcal{a}_6, \mathcal{a}_9\}$. *The mappings from* $\mathsf{hb}(\mathcal{a}_5)$, $\mathsf{hb}(\mathcal{a}_6)$ *and* $\mathsf{hb}(\mathcal{a}_9)$ *to* $\mathsf{Cl}(G_0)$ *are illustrated by Figure 3.1.*

The *scope* of an authorization over a given graph G is the set of triples in G to which the authorization is applicable. The scope is computed by the evaluation of the BGPs forming the authorization (see Definition 2.1.8 (P. 19)). Please note that the fragment defined in Definition 3.1.1 is basically used to define our access control model, and it is not meant to replace the generic SPARQL query language on RDF stores, as our approach is independent from the query language.

Definition 3.1.5 (Authorization scope) *Given a graph* $G \in \mathsf{BGP}$ *and an authorization* $\mathcal{a} \in \mathsf{Auth}$*, the scope of* $\mathcal{a}$ *on* G *is defined by the following function* $\mathsf{scope} \in \mathsf{BGP} \times \mathsf{Auth} \rightarrow \mathsf{BGP}$*:*

$$\mathsf{scope}(G)(\mathcal{a}) = \{t \in G \mid \exists \theta \in [\![\mathsf{hb}(\mathcal{a})]\!]_G . t = (\mathsf{head}(\mathcal{a}))\theta\}$$

Example 3.1.6 *Consider authorization* $\mathcal{a}_8$ *in Table 3.1, and the graph* $\mathsf{Cl}(G_0)$ *in Figure 2.4 (P. 23). The scope of* $\mathcal{a}_8$ *is computed as follows :* $\mathsf{scope}(\mathcal{a}_8) = \{et_1, it_1\}$.

3.1.2 Policy and conflict resolution function

As exemplified above, there may exist some t such that the set $\mathsf{ar}(G, \mathscr{A})(t)$ is not a singleton authorization, which can lead to policy inconsistency. 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. Also note that there may exist a triple for which the set of applicable authorizations is empty which leads to policy incompleteness. The solution to ensure that the decision function is total, is to specify a *default decision.*

To prevent us from defining many extra parameters, arbitrarily fixing some conflict resolution strategies or running into considerations on conflict resolutions, 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 Section 3.2.

Definition 3.1.7 (Policy, Conflict Resolution Function) *An (authorization) policy P is a pair $P = (\mathscr{A}, \mathsf{ch})$, satisfying the following well-formedness conditions, where $\mathscr{A}$ is a finite set of authorizations and* $\mathsf{ch} \in \mathscr{P}(\mathscr{A}) \setminus \{\emptyset\} \rightarrow \mathscr{A}$ *is a* conflict resolution function*:*

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

The subset of $\mathscr{P}(\mathsf{Auth}) \times (\mathscr{P}(\mathsf{Auth}) \rightarrow \mathsf{Auth})$ that satisfies the above well-formedness conditions is denoted by Pol*.*

The well-formedness conditions are properties which ensure that the conflict resolution functions behave well when applied to set of authorizations. The *Totality* property avoids a corner case. We explain in Section 3.2 how to enforce *default decisions* that ensure this property. The *Closedness* property guarantees that the selected authorization is taken from the input. 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.

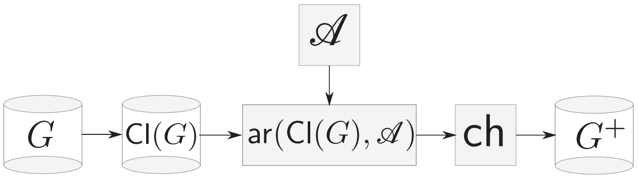


Figure 3.2: Evaluation strategy for policies

Example 3.1.8 *An example policy is $P = (\mathscr{A}, \mathsf{ch})$ where $\mathscr{A}$ is the set of authorizations in Table 3.1 and* ch *is defined as follows. For all non-empty subset $\mathscr{B}$ of $\mathscr{A}$, $\mathsf{ch}(\mathscr{B})$ is the first authorization (using syntactical order of Table 3.1) of $\mathscr{A}$ that appears in $\mathscr{B}$. Totality stems from $\mathscr{a}_9$, as it is applicable to any triple. Closedness and Monotony directly stem from the definition of* ch*.*

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

Definition 3.1.9 (Policy Evaluation, Positive Subgraph) *Given a policy $P = (\mathscr{A}, \mathsf{ch}) \in \mathsf{Pol}$ 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:*

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

As illustrated by Figure 3.2, to construct the positive subgraph, the first step consists in computing the closure of the base graph, then assigning each triple of the result with its applicable authorizations. The next step consists in applying ch on the applicable authorizations and assigning every triple with the decision of the chosen authorization. The last step consists in generating the positive subgraph which contains the triples assigned with positive effect. Algorithm 1 allows the computation of the positive subgraph, given a policy, a set of inference rules and a base graph.

Example 3.1.10 *Let us consider the policy $P = (\mathscr{A}, \mathsf{ch})$ defined in Example 3.1.8 and the graph G_0 of Figure 2.2 (P. 16). The first step consists in computing the closure of the base graph, in this case, $\mathsf{Cl}(G_0)$ shown in Figure 2.4 (P. 23). The next steps of computing every triple decision are illustrated by columns of Table 3.2. Regarding the triple $it_2 = (\texttt{:alice}\,;\texttt{:admitted}\,;\texttt{:onc})$, $\mathsf{ar}(\mathsf{Cl}(G_0), \mathscr{A})(it_2) = \{\mathscr{a}_5, \mathscr{a}_6, \mathscr{a}_9\}$. Since $\mathscr{a}_5$ is the first among authorization in Table 3.1 and its effect is $-$, we deduce that $it_2 \notin \mathsf{Cl}(G_0)^+_P$. By*

Algorithm 1 Algorithm for constructing positive sugbraph

```
Input: a base graph $G \in BGP$, an authorization policy $P = (\mathscr{A}, \mathsf{ch})$, a set of inference rules $\mathbf{R}$.
Output: a positive subgraph $G_P^+$.
function getGpos($G$, $P$, $\mathbf{R}$)
    Let $G_P^+ \leftarrow \emptyset$
    for all $t \in \mathsf{Cl}_{\mathbf{R}}(G)$ do
        Let $\mathsf{ar} \leftarrow \emptyset$
        for all $\mathscr{a} \in \mathscr{A}$ do
            if $\exists\theta$ s.t. $t = \mathsf{head}(\mathscr{a})$ then
                let $B = \mathsf{head}(\mathscr{a})\theta \cup \mathsf{hb}(\mathscr{a})\theta$
                if $[\![B]\!]_{\mathsf{Cl}_{\mathbf{R}}(G)} \neq \emptyset$ then
                    $ar \leftarrow ar \cup \{\mathscr{a}\}$
                end if
            end if
        end for
        if $(\mathsf{effect} \circ \mathsf{ch})(ar) = +$ then
            $G_P^+ \leftarrow G_P^+ \cup \{t\}$
        end if
    end for
    return $G_P^+$
end function
```

applying a similar reasoning on all triples in $\mathsf{Cl}(G_0)$*, we obtain* $\mathsf{Cl}(G_0)_P^+ = \{et_1, et_4, et_5, et_6\}$.

Table 3.2: Example of positive subgraph constitution

t	$\mathsf{ar}(\mathsf{Cl}(G_0), \mathscr{A})(t)$	$\mathsf{ch}(\mathsf{ar}(\mathsf{Cl}(G_0), \mathscr{A})(t))$	$\mathsf{effect}(\mathsf{ch}(\mathsf{ar}(\mathsf{Cl}(G_0), \mathscr{A})(t)))$
et_1	$\{\mathscr{a}_7, \mathscr{a}_8, \mathscr{a}_9\}$	$\mathscr{a}_7$	$+$
et_2	$\{\mathscr{a}_9\}$	$\mathscr{a}_9$	$-$
et_3	$\{\mathscr{a}_9\}$	$\mathscr{a}_9$	$-$
et_4	$\{\mathscr{a}_1, \mathscr{a}_9\}$	$\mathscr{a}_1$	$+$
et_5	$\{\mathscr{a}_3, \mathscr{a}_9\}$	$\mathscr{a}_3$	$+$
et_6	$\{\mathscr{a}_4, \mathscr{a}_9\}$	$\mathscr{a}_4$	$+$
it_1	$\{\mathscr{a}_2, \mathscr{a}_8, \mathscr{a}_9\}$	$\mathscr{a}_2$	$-$
it_2	$\{\mathscr{a}_5, \mathscr{a}_6, \mathscr{a}_9\}$	$\mathscr{a}_5$	$-$
it_3	$\{\mathscr{a}_9\}$	$\mathscr{a}_9$	$-$

3.1.3 Conflict resolution strategies semantics

In this section, we define properties of the ch function, related to the conflict resolution strategies. In other words, ch applies a given strategy if it satisfies the property of this strategy.

To define the DTP(PTP) properties, we need to define a partition of the set of authorizations into two sets containing the positive and negative authorizations respectively.

Definition 3.1.11 *Given a set of authorizations $\mathscr{A}$. $\forall \mathscr{B} \subseteq \mathscr{A}$. $\mathscr{B}$ can be partitioned into two sets $\mathscr{B}^-$ and $\mathscr{B}^+$ such that:*

$$\mathscr{B}^- = \{\mathscr{a} \in \mathscr{B} \mid \mathsf{effect}(\mathscr{a}) = -\}$$
$$\mathscr{B}^+ = \{\mathscr{a} \in \mathscr{B} \mid \mathsf{effect}(\mathscr{a}) = +\}$$

Example 3.1.12 *Consider the set of authorizations $\mathscr{A}$ shown in Table 3.1. Let us consider the subset of authorizations $\mathscr{B} = \{\mathscr{a}_7, \mathscr{a}_8, \mathscr{a}_9\}$. The subset of negative and positive authorizations in $\mathscr{B}$ are defined as follows: $\mathscr{B}^- = \{\mathscr{a}_7\}$, $\mathscr{B}^+ = \{\mathscr{a}_8, \mathscr{a}_9\}$*

A function ch applies the DTP(PTP) strategy if it returns a negative (positive) authorization from the input set, when the latter contains at least one negative (positive) authorization.

Definition 3.1.13 (DTP(PTP) property) *A function* ch *applies the DTP(PTP) strategy if it satisfies the DTP(PTP) property. The DTP and PTP properties are defined as follows:*

$$\texttt{DTP}\text{ property: } \forall \mathscr{B} \subseteq \mathscr{A}.\ \exists \mathscr{a} \in \mathscr{B}^- \Rightarrow \mathsf{ch}(\mathscr{B}) \in \mathscr{B}^-$$
$$\texttt{PTP}\text{ property: } \forall \mathscr{B} \subseteq \mathscr{A}.\ \exists \mathscr{a} \in \mathscr{B}^+ \Rightarrow \mathsf{ch}(\mathscr{B}) \in \mathscr{B}^+$$

Regarding the MSTP strategy, it is particularly adequate to capture exceptions in policies in a natural way. For instance, in Table 3.1, the authorization $\mathscr{a}_5$ that denies admissions to oncology service is an exception of the authorization $\mathscr{a}_6$ which allows admissions in general. According to the MSTP strategy, $\mathscr{a}_5$ should prevail over $\mathscr{a}_6$. A function ch applies the MSTP strategy if it returns an authorization that have no other more specific authorization. Semantically, an authorization $\mathscr{a}_1$ *is more specific* than $\mathscr{a}_2$ if for any given graph G, the scope of $\mathscr{a}_1$ in G is a subset of the scope of $\mathscr{a}_2$ in G, formally,

$\mathsf{scope}(G)(\mathscr{a}_1) \subseteq \mathsf{scope}(G)(\mathscr{a}_2)$. Syntactically, an authorization $\mathscr{a}_1$ is more specific than authorization $\mathscr{a}_2$ when the underlying graph pattern of $\mathscr{a}_2$ can be matched to the one of $\mathscr{a}_1$ with the restriction that the head of $\mathscr{a}_2$ is mapped to the head of $\mathscr{a}_1$.

Definition 3.1.14 *We say that an authorization $\mathscr{a}_1$ is more specific than authorization $\mathscr{a}_2$ denoted by $\mathscr{a}_1 \sqsubseteq_{MS} \mathscr{a}_2$ if $\exists\theta.\, \mathsf{hb}(\mathscr{a}_2)\theta \subseteq \mathsf{hb}(\mathscr{a}_1) \wedge \mathsf{head}(\mathscr{a}_2)\theta = \mathsf{head}(\mathscr{a}_1)$.*

Example 3.1.15 *Consider authorizations $\mathscr{a}_5$ and $\mathscr{a}_6$ shown in Table 3.1. There exists a substitution mapping θ between $\mathscr{a}_5$ and $\mathscr{a}_6$ s.t. $\theta = \{?\mathtt{p} \mapsto ?\mathtt{p}, ?\mathtt{s} \mapsto ?\mathtt{s}\}$ and $\mathsf{hb}(\mathscr{a}_6)\theta \subseteq \mathsf{hb}(\mathscr{a}_5) \wedge \mathsf{head}(\mathscr{a}_6)\theta = \mathsf{head}(\mathscr{a}_5)$, hence $\mathscr{a}_6 \sqsubseteq_{MS} \mathscr{a}_5$;*

After defining the specificity relation, we can now check if a function ch satisfies the `MSTP` property, *i.e.* when it returns one of the smallest authorizations with respect to $\sqsubseteq_{\mathtt{MS}}$.

Definition 3.1.16 (`MSTP`) property) *A function ch applies the `MSTP` strategy if it satisfies the following property:*

Let $\sqsubseteq_{\mathtt{MS}}$ be the specificity relation between authorizations
`MSTP` property: $\forall \mathscr{B} \subseteq \mathscr{A}.\, \mathsf{ch}(\mathscr{B}) = \mathscr{a} \Rightarrow (\forall \mathscr{a}' \in \mathscr{B}.\, \mathscr{a}' \sqsubseteq_{\mathtt{MS}} \mathscr{a} \Rightarrow \mathscr{a}' \sqsupseteq_{\mathtt{MS}} \mathscr{a})$

3.2 Building Policies

In this section, we illustrate the applicability of policies as defined in Definition 3.1.7 by showing how to construct ch functions for applying conflict resolution strategies known in the literature. Note that our ch function is responsible for resolving conflicts between authorizations, as well as applying the default strategy.

3.2.1 Default Strategy

A default strategy is a decision that is selected when no other authorization is applicable, that is when $\mathsf{ar}(G, \mathscr{A})(t) = \emptyset$. Such a default strategy can either be *deny by default* or *permit by default.* In order to respect the *Totality* well-formedness condition of Definition 3.1.7, we cannot simply apply a *default decision*. However, we have to identify a *default authorization* called *universal authorization* denoted by $\mathscr{a}_u$, that is applicable to any triple. $\mathscr{a}_u$ is added to the set of authorizations in order to achieve *Totality*. In fact, the triples that have no applicable authorizations but $\mathscr{a}_u$, will have a decision that is equal to the effect of $\mathscr{a}_u$. Hence the effect of $\mathscr{a}_u$ plays the role of the default decision of classical approaches. An authorization is applicable to any triple if it has three different variables in the head, and an empty body.

Definition 3.2.1 *An authorization* $\mathscr{a}_u \in \mathsf{Auth}$ *is called universal if* $\mathscr{a}_u = \mathtt{\mathit{GRANT}}/\mathtt{\mathit{DENY}}\,(\mathsf{s}\,;\mathsf{p}\,;\mathsf{o})$ *such that* $\mathsf{s}, \mathsf{p}, \mathsf{o} \in \mathsf{V}$ *and* $\mathsf{s} \neq \mathsf{p}, \mathsf{p} \neq \mathsf{o}, \mathsf{s} \neq \mathsf{o}$.

The following lemma shows that the *Totality* condition can be ensured by adding a universal authorization.

Lemma 3.2.2 *Let* $\mathscr{A}$ *be a finite subset of* Auth*. The following statements are equivalent:*

$$\forall G. \forall t \in G.\, \mathsf{ar}(G, \mathscr{A})(t) \neq \emptyset$$
$$\exists \mathscr{a}_u \in \mathscr{A}. \forall G. \forall t \in G. \mathscr{a}_u \in \mathsf{ar}(G, \mathscr{A})(t)$$

Proof One direction is straightforward, if $\mathscr{a}_u \in \mathscr{A}$ is such that $\mathscr{a}_u \in \mathsf{ar}(G, \mathscr{A})(t)$, then $\mathsf{ar}(G, \mathscr{A})(t) \neq \emptyset$.

For the opposite direction, consider a triple $t^e = (?\mathsf{s}^\mathsf{e}\,;?\mathsf{p}^\mathsf{e}\,;?\mathsf{o}^\mathsf{e})$ with fresh variables not already used in $\mathscr{A}$, that is, t^e is such that for all authorizations $\mathscr{a} \in \mathscr{A}$, it is the case that $\forall(?\mathsf{s}; ?\mathsf{p}; ?\mathsf{o}) \in \mathsf{hb}(\mathscr{a}).?\mathsf{s} \neq ?\mathsf{s}^\mathsf{e} \wedge ?\mathsf{p} \neq ?\mathsf{p}^\mathsf{e} \wedge ?\mathsf{p} \neq ?\mathsf{p}^\mathsf{e}$. Such a triple t^e always exists because there is an infinite set V of variables but only finitely many appear in $\mathscr{A}$. Consider the graph $G^e = \{t^e\}$, by hypothesis $\mathsf{ar}(G^e, \mathscr{A})(t^e)$ is not empty, so consider an authorization $\mathscr{a}_u$ in this set.

We show that this authorization is universal. By Definition 3.1.3, we know that there exists some $\eta \in [\![\mathsf{hb}(\mathscr{a}_u)]\!]_G$ with $t^e = (\mathsf{head}(\mathscr{a}_u))\eta$. Thus, by Definition 2.1.8 (P. 19), we have that $(\mathsf{hb}(\mathscr{a}_u))\eta = \{t^e\}$ because $\mathsf{hb}(\mathscr{a}_u)$ cannot be empty. Let G^a an arbitrary graph and $t^a = (?\mathsf{s}^\mathsf{a}\,;?\mathsf{p}^\mathsf{a}\,;?\mathsf{o}^\mathsf{a}) \in G^a$. Consider the substitution η' that maps t^e to t^a defined by $(?\mathsf{s}^\mathsf{e})\eta' = ?\mathsf{s}^\mathsf{a}$, $(?\mathsf{p}^\mathsf{e})\eta' = ?\mathsf{p}^\mathsf{a}$. By construction, $(\mathsf{hb}(\mathscr{a}_u))\eta\eta' = \{t^a\}$ and $t^e = (\mathsf{head}(\mathscr{a}_u))\eta\eta'$, thus $\mathscr{a}_u \in \mathsf{ar}(G^a, \mathscr{A})(t^a)$. $\blacksquare$

For instance, the default strategy in Table 3.1 is given by authorization $\mathscr{a}_9$ which effect is `DENY`. Hence a closed policy. Note that there may be several different universal authorizations in the set $\mathscr{A}$. Therefore, conflicts will be systematically triggered. Even though it is formally possible to have several universal authorizations, we can assume that such authorization is *unique*.

Assumption 3.2.3 *Given a policy $P = (\mathscr{A}, \mathsf{ch})$.The set of authorizations $\mathscr{A}$ contains a unique universal authorization.*

Note that the addition of a default rule at the end of a rule set is standard practice in firewall policies. Moreover, following previous discussion on the total syntactical order, this authorization should be naturally appended to the end of the set of authorizations. More generally, given a total order $\preccurlyeq$ on $\mathscr{A}$, the universal authorization should be the *maximum* element of $\mathscr{A}$ with respect to $\preccurlyeq$. Doing so, it wont be selected unless it is the only choice. This reflects the semantics of the default strategy which only applies when there is no other applicable authorization.

3.2.2 First Applicable strategy

First of all, we notice that if there exists a *total order* denoted by $\preccurlyeq$ over a set of authorizations $\mathscr{A}$, we can apply the `FA` satrategy by constructing a conflict resolution function ch that selects the *minimum* element from a subset $\mathscr{B} \subseteq \mathscr{A}$ ordered by $\preccurlyeq$.

Definition 3.2.4 *Given a set of authorizations $\mathscr{A}$ and a total order $\preccurlyeq$ on $\mathscr{A}$. For any $\mathscr{B} \subseteq \mathscr{A}$, the minimum authorization of $\mathscr{B}$ with respect to $\preccurlyeq$ is defined as follows:*

$$\mathsf{min}_{\preccurlyeq}(\mathscr{B}) = \mathscr{a} \in \mathscr{B} \mid \forall \mathscr{a}' \in \mathscr{A} \,.\, \mathscr{a} \preccurlyeq \mathscr{a}'$$

Lemma 3.2.5 *Given a set of authorizations $\mathscr{A}$ and a total order $\preccurlyeq$ on $\mathscr{A}$, $P = (\mathscr{A}, \mathsf{min}_{\preccurlyeq})$ is a well-formed policy.*

Proof We prove that $P = (\mathscr{A}, \mathsf{min}_{\preccurlyeq})$ satisfies the well-formedness conditions.

For *Totality*, by assumption 3.2.3, there exists a universal rule $\mathscr{a}_u \in \mathscr{A}$. *Closedness* and the *Monotony* are satisfied by construction of $\mathsf{min}_{\preccurlyeq}$. $\blacksquare$

There are several ways to equip $\mathscr{A}$ 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. 3.1, 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.

Example 3.2.6 $\mathsf{min}_{\preccurlyeq}$ *represents the* ch *function defined in Example 3.1.8, where the authorizations are ordered syntactically.*

Note that given a policy $P = (\mathscr{A}, \mathsf{ch})$, we can construct a total order $\preccurlyeq_{\mathsf{ch}}$ on $\mathscr{A}$ with ch. From an implementation point of view, constructing a total order from ch could improve performance of the access control enforcement. The idea is to compute the total order $\preccurlyeq_{\mathsf{ch}}$ during policy design time, and replace the original ch by $\mathsf{min}_{\preccurlyeq_{\mathsf{ch}}}$.

Lemma 3.2.7 *Given a policy* $P = (\mathscr{A}, \mathsf{ch})$*, then we can construct a total order* $\preccurlyeq_{\mathsf{ch}}$ *on* $\mathscr{A}$ *with* ch *function s.t.* $\forall a_1, a_2 \in \mathscr{A} . a_1 \preccurlyeq_{\mathsf{ch}} a_2 \iff \mathsf{ch}(\{a_1, a_2\}) = a_1$.

Proof We prove that $\preccurlyeq_{\mathsf{ch}}$ is reflexive, antisymmetric, transitive and total.

- Reflexivity: Since $\mathsf{ch}(\{a, a\}) = a$ then $a \preccurlyeq_{\mathsf{ch}} a$. Thus $\preccurlyeq_{\mathsf{ch}}$ is reflexive.
- Antisymmetry: If $a_1 \preccurlyeq_{\mathsf{ch}} a_2$ then $\mathsf{ch}(\{a_1, a_2\}) = a_1$, hence if $a_1 \neq a_2$ then $\mathsf{ch}(\{a_2, a_1\}) \neq a_2$, which means that $a_2 \not\preccurlyeq_{\mathsf{ch}} a_1$. Thus $\preccurlyeq_{\mathsf{ch}}$ is antisymmetric.
- Transitivity: We prove that if $a_1 \preccurlyeq_{\mathsf{ch}} a_2$ and $a_2 \preccurlyeq_{\mathsf{ch}} a_3$ then $a_1 \preccurlyeq_{\mathsf{ch}} a_3$.
  If $\mathsf{ch}(\{a_1, a_2, a_3\}) = a_3$ then $\mathsf{ch}(\{a_2, a_3\}) = a_3$ from *Monotony* condition. Contradiction with $a_2 \preccurlyeq_{\mathsf{ch}} a_3$.
  If $\mathsf{ch}(\{a_1, a_2, a_3\}) = a_2$ then $\mathsf{ch}(\{a_1, a_2\}) = a_2$ from *Monotony* condition. Contradiction with $a_1 \preccurlyeq_{\mathsf{ch}} a_2$.
  Since $\mathsf{ch}(\{a_1, a_2, a_3\}) \neq a_3$ and $\mathsf{ch}(\{a_1, a_2, a_3\}) \neq a_2$ then $\mathsf{ch}(\{a_1, a_2, a_3\}) = a_1$ from ch *Closedness*. Hence $\mathsf{ch}(\{a_1, a_3\}) = a_1$ from ch *Monotony*. Which means that $a_1 \preccurlyeq_{\mathsf{ch}} a_3$. Thus $\preccurlyeq_{\mathsf{ch}}$ is transitive.
- Totality: By ch *Closedness*, $\forall a_1, a_2 \in \mathscr{A} . \mathsf{ch}(\{a_1, a_2\}) = a_1$ or $\mathsf{ch}(\{a_1, a_2\}) = a_2$. Hence, $a_1 \preccurlyeq_{\mathsf{ch}} a_2$ or $a_2 \preccurlyeq_{\mathsf{ch}} a_1$. Thus $\preccurlyeq_{\mathsf{ch}}$ is total.

The following proposition shows that there exists a bijection between total orders and choice functions.

Proposition 3.2.8 *Given a policy* $P = (\mathscr{A}, \mathsf{ch})$*, there exists a bijection between* $\preccurlyeq_{\mathsf{ch}}$ *and* ch

Proof Let $P = (\mathscr{A}, \mathsf{ch})$ be a policy.

Let $\preccurlyeq_{\mathsf{ch}}$ be a binary relation s.t $\forall \mathscr{a}_1, \mathscr{a}_2 \in \mathscr{A} . \mathscr{a}_1 \preccurlyeq_{\mathsf{ch}} \mathscr{a}_2 \iff \mathsf{ch}(\mathscr{a}_1, \mathscr{a}_2) = \mathscr{a}_1$.

First we prove that $\mathsf{min}_{\preccurlyeq_{\mathsf{ch}}}(\mathscr{A}) = \mathsf{ch}(\mathscr{A})$.

If $\mathsf{min}_{\preccurlyeq_{\mathsf{ch}}}(\mathscr{A}) = \mathscr{a} \in \mathscr{A}$ then $\forall \mathscr{a}' \in \mathscr{A} . \mathscr{a} \preccurlyeq_{\mathsf{ch}} \mathscr{a}'$ (by $\preccurlyeq_{\mathsf{ch}}$ definition).

Let $\mathscr{b} = \mathsf{ch}(\mathscr{A})$. We have $\mathscr{b} \in \mathscr{A}$, hence $\mathscr{a} \preccurlyeq_{\mathsf{ch}} \mathscr{b}$, which means that $\mathsf{ch}(\mathscr{a}, \mathscr{b}) = \mathscr{a}$. Since $\{\mathscr{a}, \mathscr{b}\} \subseteq \mathscr{A}$, then, by ch *Monotony*, $\mathsf{ch}(\mathscr{a}, \mathscr{b}) = \mathscr{b} = \mathsf{ch}(\mathscr{A})$, hence $\mathscr{a} = \mathscr{b}$.

Let $\preccurlyeq$ be a total order. Let $\mathsf{min}_{\preccurlyeq}$ the choice function associated to $\preccurlyeq$. Let $\preccurlyeq_{\mathsf{min}_{\preccurlyeq}}$ a total order constructed from $\mathsf{min}_{\preccurlyeq}$ s.t $\forall \mathscr{a}_1, \mathscr{a}_2 \in \mathscr{A} . \mathscr{a}_1 \preccurlyeq_{\mathsf{min}_{\preccurlyeq}} \mathscr{a}_2 \iff \mathscr{a}_1 = \mathsf{min}_{\preccurlyeq}(\{\mathscr{a}_1, \mathscr{a}_2\})$.

We prove that $\preccurlyeq_{\mathsf{min}_{\preccurlyeq}} \iff \preccurlyeq$

$\mathscr{a}_1 \preccurlyeq_{\mathsf{min}_{\preccurlyeq}} \mathscr{a}_2 \iff \mathsf{min}_{\preccurlyeq}(\{\mathscr{a}_1, \mathscr{a}_2\}) = \mathscr{a}_1 \iff \mathscr{a}_1 \preccurlyeq \mathscr{a}_2$

3.2.3 Precedence Strategies

As we mentioned in Chapter 2 (P. 13), the `DTP` strategy resolves conflicts by stating that the negative authorizations prevail over the positive ones; the `PTP` strategy being its dual. The idea to capture the `DTP` (resp. `PTP`) strategy is to transform a policy $P = (\mathscr{A}, \mathsf{ch})$ into a policy $P^{-} = (\mathscr{A}, \mathsf{ch}^{-})$ where ch^{-} privileges negative (resp. positive) effects. Considering the previous discussion on default policies, we assume that there is a unique universal authorization $\mathscr{a}_u \in \mathscr{A}$. As $\mathscr{a}_u$ is assumed to be a default authorization, we require that $\mathscr{B} \setminus \{\mathscr{a}_u\} = \emptyset$ if and only if $\mathsf{ch}(\mathscr{B}) = \mathscr{a}_u$. Remind that $\mathscr{B}^{-}$ (resp. $\mathscr{B}^{+}$) is the subset of $\mathscr{B}$ with a negative (resp. positive) effect. With $\mathscr{B} \subseteq \mathscr{A}$, the ch^{-} function is formally defined as follows:

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

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 3.2.9 (Correctness of P^{-}) *Given $P = (\mathscr{A}, \mathsf{ch})$ a policy according to Definition 3.1.7 with a unique universal authorization $\mathscr{a}_u \in \mathscr{A}$ such that*

$\forall \mathscr{B} \subseteq \mathscr{A}.\mathsf{ch}(\mathscr{B}) = \mathscr{a}_u \Rightarrow \mathscr{B} \setminus \{\mathscr{a}_u\} = \emptyset$*, the structure* $P^- = (\mathscr{A}, \mathsf{ch}^-)$ *is a well-formed policy.*

Proof Firstly, we remark that $\mathscr{B}^- \setminus \{\mathscr{a}_u\} = (\mathscr{B} \setminus \{\mathscr{a}_u\})^-$ and that $\mathscr{B}^- \cup \mathscr{B}^+ = \mathscr{B}$. Secondly, we note that function ch^- is properly defined because pairwise conjunctions of conditions (1), (2) and (3) are unsatisfiable and the disjunction of these formulas is a tautology. We show that $P = (\mathscr{A}, \mathsf{ch}^-)$ satisfies the well-formedness conditions.

Totality By assumption 3.2.3, there exists a universal rule $\mathscr{a}_u \in \mathscr{A}$.

Closedness This property is guaranteed because the original ch function is assumed to satisfy the *Closedness* property.

Monotony Let $\mathsf{ch}^-(\mathscr{B}) = \mathscr{a}$ for some $\mathscr{B} \subseteq \mathscr{A}$ and let $\mathscr{B}' \subseteq \mathscr{B}$ with $\mathscr{a} \in \mathscr{B}'$. We have to show that $\mathsf{ch}^-(\mathscr{B}') = \mathscr{a}$. If $\mathscr{a} = \mathscr{a}_u$, then $\mathscr{B} \setminus \{\mathscr{a}_u\} = \emptyset$ by hypothesis so $\mathscr{B}' \setminus \{\mathscr{a}_u\} = \emptyset$ as well, thus $\mathsf{ch}^-(\mathscr{B}') = \mathsf{ch}^-(\mathscr{B}) = \mathscr{a}_u$. So we assume now that $\mathscr{a} \neq \mathscr{a}_u$, this implies that $\mathscr{a} \in \mathscr{B} \setminus \{\mathscr{a}_u\}$ by the *Closedness* property of the ch function, thus $\mathscr{B} \setminus \{\mathscr{a}_u\} \neq \emptyset$. Moreover, $\mathsf{ch}(\mathscr{B} \setminus \{\mathscr{a}_u\}) = \mathsf{ch}(\mathscr{B})$ by the *Monotony* condition. We analyse how $\mathscr{a}$ was obtained in the first place.

case $\mathscr{B}^- \setminus \{\mathscr{a}_u\} \neq \emptyset$**.** If $\mathscr{B}'^- \setminus \{\mathscr{a}_u\} \neq \emptyset$ holds, we have that $\mathsf{ch}^-(\mathscr{B}') = \mathsf{ch}(\mathscr{B}'^- \setminus \{\mathscr{a}_u\})$ by the definition of ch^-. Then $\mathsf{ch}(\mathscr{B}'^- \setminus \{\mathscr{a}_u\}) = \mathsf{ch}(\mathscr{B}^- \setminus \{\mathscr{a}_u\})$ by the *Monotony* of the ch function. Finally, $\mathsf{ch}(\mathscr{B}^- \setminus \{\mathscr{a}_u\}) = \mathsf{ch}((\mathscr{B} \setminus \{\mathscr{a}_u\})^-) = \mathsf{ch}(\mathscr{B} \setminus \{\mathscr{a}_u\})$ by *Monotony* again, so $\mathsf{ch}(\mathscr{B} \setminus \{\mathscr{a}_u\}) = \mathsf{ch}(\mathscr{B}) = \mathscr{a} = \mathsf{ch}^-(\mathscr{B}')$.

Otherwise, we have $\mathscr{B}'^- \setminus \{\mathscr{a}_u\} = \emptyset$. Assumption $\mathscr{a} \neq \mathscr{a}_u$ implies that case (3) is ruled out, so $\mathscr{a}$ has to be positive by case (2). By the *Closedness* property of ch, $\mathscr{a} \in \mathscr{B}^- \setminus \{\mathscr{a}_u\}$ so $\mathscr{a}$ is negative, a contradiction.

case $\mathscr{B}^- \setminus \{\mathscr{a}_u\} = \emptyset \wedge \mathscr{B}^+ \setminus \{\mathscr{a}_u\} \neq \emptyset$**.** Note that $\mathscr{B}'^- \setminus \{\mathscr{a}_u\} = \emptyset$ as well because $\mathscr{B}' \subseteq \mathscr{B}$. If $\mathscr{B}'^+ \setminus \{\mathscr{a}_u\} \neq \emptyset$ holds, then we obtain the equalities $\mathsf{ch}^-(\mathscr{B}') = \mathsf{ch}(\mathscr{B}'^+ \setminus \{\mathscr{a}_u\}) = \mathsf{ch}(\mathscr{B}^+ \setminus \{\mathscr{a}_u\}) = \mathsf{ch}^-(\mathscr{B} \setminus \{\mathscr{a}_u\})$, similarly to the previous case. Otherwise, $(\mathscr{B}' \setminus \{\mathscr{a}_u\})^+ = \emptyset$, a contradiction with $\mathscr{a} \in \mathscr{B}'$.

case $\mathscr{B} \setminus \{\mathscr{a}_u\} = \emptyset$ It is immediate because $\mathsf{ch}^-(\mathscr{B}') = \mathscr{a}_u = \mathsf{ch}^-(\mathscr{B})$. ∎

The following lemma ensures that ch^- applies the `DTP` strategy.

Lemma 3.2.10 (DTP strategy application) *Given a policy $P^- = (\mathscr{A}, \mathsf{ch}^-)$, ch^- applies the* `DTP` *strategy.*

Proof We prove that ch^- satisfies the `DTP` property (see Definition 3.1.13).

$\forall \mathscr{B} \subseteq \mathscr{A}$. if $\exists \mathscr{a} \in \mathscr{B}^-$ s.t. $\mathscr{a} \neq \mathscr{a}_u$ then $\mathscr{B}^- \setminus \{\mathscr{a}_u\} \neq \emptyset$, hence, by ch^- definition, $\mathsf{ch}^-(\mathscr{B}) = \mathsf{ch}(\mathscr{B}^- \setminus \{\mathscr{a}_u\})$. Which means that $\mathsf{ch}^-(\mathscr{B}) \in \mathscr{B}^-$. Thus ch^- satisfies the `DTP` property. ■

Similarly, Lemmas 3.2.9 and 3.2.10 apply to P^+ and ch^+ respectively.

Example 3.2.11 *Consider the graph $\mathsf{Cl}(G_0)$ shown in Figure 2.4 (P. 23) and the set of authorizations $\mathscr{A}$ shown in Table 3.1. Let us consider the authorizations applicable to triple et_1, that is $\mathsf{ar}(\mathsf{Cl}(G_0), \mathscr{A})(et_1) = \{\mathscr{a}_7, \mathscr{a}_8, \mathscr{a}_9\}$. If we consider the ch given in Example 3.1.8, that is, the syntactical order, authorization $\mathscr{a}_7$, a positive one, is selected. However, with the* `DTP` *construction, we have that $\mathsf{ch}^-(\{\mathscr{a}_7, \mathscr{a}_8, \mathscr{a}_9\}) = \mathsf{ch}(\{\mathscr{a}_8\}) = \mathscr{a}_8$.*

3.2.4 Most Specific Takes Precedence (MSTP)

We showed in Section 3.1.3 that the `MSTP` property is captured by a binary relation $\sqsubseteq_{\mathtt{MS}}$ defined by substitutions. Clearly, the identity substitution makes $\sqsubseteq_{\mathtt{MS}}$ relation reflexive and composition of substitution makes it transitive. Therefore, it is a *preorder (quasiorder)*. Note that $\sqsubseteq_{\mathtt{MS}}$ being a preorder, it is not anti-symmetric. Indeed, we may have two different authorizations with different effects $\mathscr{a}_1$ and $\mathscr{a}_2$ where $\mathscr{a}_1 \sqsubseteq_{\mathtt{MS}} \mathscr{a}_2$ and $\mathscr{a}_2 \sqsubseteq_{\mathtt{MS}} \mathscr{a}_1$. Chosing the most specific authorization amounts to select the smallest authorization w.r.t $\sqsubseteq_{\mathtt{MS}}$. Since $\sqsubseteq_{\mathtt{MS}}$ is a preorder, then the smallest authorizations may be more than one. As shown by Lemma 3.2.5, given a total $\mathsf{min}_{\preccurlyeq}$ order, we can construct a policy $P = (\mathscr{A}, \mathsf{min}_{\preccurlyeq})$. The idea is to construct a total order from the preorder $\sqsubseteq_{\mathtt{MS}}$ to make a ch function that applies `MSTP`. First, we consider two equivalent authorizations as incomparable in order to make a *partial order* $\sqsubseteq_{\mathtt{MSTP}}$ from the preorder $\sqsubseteq_{\mathtt{MS}}$.

Definition 3.2.12 *Given a set of authorizations $\mathscr{A}$. $\sqsubseteq_{\mathtt{MSTP}}$ is a binary relation over $\mathscr{A}$ defined as follows:*

- $\forall \mathscr{a}_1 \in \mathscr{A}, \forall \mathscr{a}_2 \in \mathscr{A}.\ \mathscr{a}_1 \neq \mathscr{a}_2.\ \mathscr{a}_1 \sqsubseteq_{\mathtt{MSTP}} \mathscr{a}_2$ *if* $\mathscr{a}_1 \sqsubseteq_{\mathtt{MS}} \mathscr{a}_2$ *and* $\mathscr{a}_2 \not\sqsubseteq_{\mathtt{MS}} \mathscr{a}_1$
- $\forall \mathscr{a} \in \mathscr{A} . \mathscr{a} \sqsubseteq_{\mathtt{MSTP}} \mathscr{a}$

Lemma 3.2.13 *$\sqsubseteq_{\mathit{MSTP}}$ is a partial order.*

Proof We prove that $\sqsubseteq_{\texttt{MSTP}}$ is reflexive, anti-symmetric and transitive.

- *Reflexivity* : By Definition 3.2.12, $\sqsubseteq_{\texttt{MSTP}}$ is reflexive.
- *Transitivity* : We prove that if $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_2$ and $\mathscr{a}_2 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_3$ then $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_3$.
  If $\mathscr{a}_1 = \mathscr{a}_2$ or $\mathscr{a}_2 = \mathscr{a}_3$, then transitivity is trivial.
  By Definition 3.2.12
  $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_2 \Rightarrow \mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_2$ and $\mathscr{a}_2 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_1$
  $\mathscr{a}_2 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_3 \Rightarrow \mathscr{a}_2 \sqsubseteq_{\texttt{MS}} \mathscr{a}_3$ and $\mathscr{a}_3 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_2$
  We want to prove that $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_3$ *i.e.* $\mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_3$ and $\mathscr{a}_3 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_1$
  Since $\mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_2$ and $\mathscr{a}_2 \sqsubseteq_{\texttt{MS}} \mathscr{a}_3$ then $\mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_3$ by $\sqsubseteq_{\texttt{MS}}$ transitivity.
  Now we prove that $\mathscr{a}_3 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_1$. For the sake of the contradiction, suppose that $\mathscr{a}_3 \sqsubseteq_{\texttt{MS}} \mathscr{a}_1$. Since $\mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_2$ then $\mathscr{a}_3 \sqsubseteq_{\texttt{MS}} \mathscr{a}_2$ by $\sqsubseteq_{\texttt{MS}}$ transitivity. Which is a contradiction with $\mathscr{a}_3 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_2$, hence $\mathscr{a}_3 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_1$. Since $\mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_3$ and $\mathscr{a}_3 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_1$ then $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_3$ by Definition 3.2.12.
- *Anti-symmetry* : We prove that if $\mathscr{a}_1 \neq \mathscr{a}_2$ and $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_2$ then $\mathscr{a}_2 \not\sqsubseteq_{\texttt{MSTP}} \mathscr{a}_1$
  By Definition 3.2.12
  $\mathscr{a}_1 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_2 \Rightarrow \mathscr{a}_1 \sqsubseteq_{\texttt{MS}} \mathscr{a}_2$ and $\mathscr{a}_2 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_1$
  For the sake of the contradiction, suppose that $\mathscr{a}_2 \sqsubseteq_{\texttt{MSTP}} \mathscr{a}_1$ *i.e.* $\mathscr{a}_2 \sqsubseteq_{\texttt{MS}} \mathscr{a}_1$ and $\mathscr{a}_1 \not\sqsubseteq_{\texttt{MS}} \mathscr{a}_2$, contradiction. Hence $\mathscr{a}_2 \not\sqsubseteq_{\texttt{MSTP}} \mathscr{a}_1$.

Hence $\sqsubseteq_{\texttt{MSTP}}$ is partial order $\blacksquare$

After making a partial order $\sqsubseteq_{\texttt{MSTP}}$ from the preorder $\sqsubseteq_{\texttt{MS}}$, we can now construct a total order $\preccurlyeq_{\texttt{MSTP}}$ using the partial order $\sqsubseteq_{\texttt{MSTP}}$ and the total lexical order $\preccurlyeq_{\texttt{LEX}}$ already defined by the administrator. $\preccurlyeq_{\texttt{MSTP}}$ represents a *topological sort* (*i.e.* *linear extension*) of the partially ordered authorizations. Many algorithms could be used for the topological sort, such as Kahn's Algorithm [Kahn 1962]. We adapted Kahn's Algorithm to make it deterministic, where instead of selecting a random authorization, we select the smallest one with respect to $\preccurlyeq_{\texttt{LEX}}$. Algorithm 2 takes as parameters, the total order $\preccurlyeq_{\texttt{LEX}}$ and $\mathsf{H}^{\mathscr{A}}_{\sqsubseteq}$ a Directed Acyclic Graph (DAG) where the nodes represent authorizations in $\mathscr{A}$, and edges represent the irreflexive related authorizations with $\sqsubseteq$, *i.e.* ($\mathscr{a} \sqsubseteq \mathscr{b}$ and $\mathscr{a} \neq \mathscr{b}$) $\Rightarrow \mathscr{a} \rightarrow \mathscr{b}$.

Algorithm 2 starts by inserting all authorizations without incoming edges into a set S. Then it starts a loop where the smallest authorization $\mathscr{a}$ in S

w.r.t $\preccurlyeq_{\mathtt{LEX}}$ is inserted into L at an indice equal to the current iteration. $\mathscr{a}$ is then removed from S and all edges outgoing from it are removed as well. Next, the new authorizations with no incoming edges are inserted to S. The loop stops when S becomes empty. Finally if there are remaining edges in the graph, then it contains cycles, otherwise the Algorithm returns the total order $\preccurlyeq^{\mathscr{A}}_{\sqsubseteq}$ representing a linear extension of $\sqsubseteq$.

Algorithm 2 Algorithm for constructing a total order from partially ordered authorizations

```
Input: a graph $\mathsf{H}^{\mathscr{A}}_{\sqsubseteq}$ of authorizations ordered by $\sqsubseteq$, and the total syntactical
    order $\preccurlyeq_{\mathtt{LEX}}$.
Output: a total order relation $\preccurlyeq^{\mathscr{A}}_{\sqsubseteq}$.
 1: Let $L \leftarrow$ an Array that will contain the sorted authorizations, with indices
    from $1..|\mathscr{A}|$
 2: Let $S \leftarrow$ the set of all authorizations with no incoming edges
 3: Let $iteration \leftarrow 0$
 4: while $S$ is not empty do
 5:     $iteration \leftarrow iteration + 1$
 6:     Let $\mathscr{a} = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(S)$
 7:     $L[iteration] \leftarrow \mathscr{a}$
 8:     Remove $\mathscr{a}$ from $S$
 9:     for all $\mathscr{a}'$ with an edge $e$ from $\mathscr{a}$ to $\mathscr{a}'$ do
10:         remove $e$ from $\mathsf{H}^{\mathscr{A}}_{\sqsubseteq}$
11:         if $\mathscr{a}'$ has no other incoming edges then
12:             insert $\mathscr{a}'$ into $S$
13:         end if
14:     end for
15: end while
16: if $\mathsf{H}^{\mathscr{A}}_{\sqsubseteq}$ has edges then
17:     Error, the graph contains a cycle, and the topological sort is not pos-
    sible
18: else
19:     Let $\preccurlyeq^{\mathscr{A}}_{\sqsubseteq}$ be the following binary relation: $L[i] \preccurlyeq^{\mathscr{A}}_{\sqsubseteq} L[j] \iff i \leq j$
20:     Return $\preccurlyeq^{\mathscr{A}}_{\sqsubseteq}$
21: end if
```

The following lemma shows that at each iteration of the loop between lines 4 and 15, the next selected authorization is the smallest one w.r.t $\preccurlyeq_{\mathtt{LEX}}$ from the rest of the not processed authorizations.

Lemma 3.2.14 *Let ns be the set of the not sorted authorizations defined as $ns = \{\mathscr{a} \mid \mathscr{a} \in \mathscr{A} \wedge \mathscr{a} \notin L\}$*

Let ns_i and S_i be the values of ns and S respectively at the beginning of the i^{th} iteration of the loop between lines 4 and 15.

The following holds:
For each iteration i, authorization $\mathscr{a}'' = \mathsf{min}_{\preccurlyeq_{\mathit{LEX}}}(ns_i)$ has no incoming edges, and it is the next authorization that will be inserted into L, i.e. $L[i] = \mathsf{min}_{\preccurlyeq_{\mathit{LEX}}}(ns_i)$.

Proof First we prove that $\mathscr{a}'' = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_i)$ has no incoming edges. Suppose that $\mathscr{a}''$ has an incoming edge from $\mathscr{b}$. $\mathscr{b}$ is not in L, otherwise the edge $\mathscr{b} \rightarrow \mathscr{a}''$ would have been removed by Line 10 during some previous iteration. Hence $\mathscr{b} \in ns_i$. Since there is an edge $\mathscr{b} \rightarrow \mathscr{a}$ then $\mathscr{b} \sqsubseteq \mathscr{a}''$, thus $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{a}''$ since $\preccurlyeq_{\mathtt{LEX}}$ is a linear extension of $\sqsubseteq$. Since we do not consider atomic loops, then $\mathscr{a}'' \neq \mathscr{b}$. Since $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{a}''$ then $\mathscr{a}'' \neq \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_i)$ which contradicts the hypothesis.

We prove that the next authorization that will be inserted into L is $\mathscr{a}'' = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_i)$. Suppose that $\mathscr{a}''$ is not the next authorization to be inserted to L, hence $\exists \mathscr{b} \in S_i \subseteq ns_i$ s.t. $\mathscr{b} = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(S_i)$. Since $\mathscr{b} \in ns_i$ and $\mathscr{a}'' = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_i)$ then $\mathscr{a}'' \preccurlyeq_{\mathtt{LEX}} \mathscr{b}$.

$\mathscr{a}'' = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_i)$ means that $\mathscr{a}''$ has no incoming edges, hence $\mathscr{a}'' \in S_i$ (Lines 11 and 12). Since $\mathscr{b} = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(S_i)$ and $\mathscr{a}'' \in S_i$ then $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{a}''$.

Since $\mathscr{a}'' \preccurlyeq_{\mathtt{LEX}} \mathscr{b}$ and $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{a}''$ then $\mathscr{a}'' = \mathscr{b}$.

Note that if the total order $\preccurlyeq_{\mathtt{LEX}}$ is already a linear extension of the partial order $\sqsubseteq$, then Algortithm 2 will generate a total order that is equal to $\preccurlyeq_{\mathtt{LEX}}$, which is showed by the following lemma.

Lemma 3.2.15 *If $\preccurlyeq_{\mathit{LEX}}$ is a linear extension of $\sqsubseteq_{\mathit{MSTP}}$ then $\preccurlyeq_{\mathit{LEX}} = \preccurlyeq^{\mathscr{A}}_{\sqsubseteq}$*

Proof We show that at the end of Algorithm 2 :
If $i \leq j$ then $L[i] \preccurlyeq_{\mathtt{LEX}} L[j]$, and if $L[i] \preccurlyeq_{\mathtt{LEX}} L[j]$ then $i \leq j$.

We prove that if $i \leq j$ then $L[i] \preccurlyeq_{\mathtt{LEX}} L[j]$. Let $\mathscr{b} = L[i]$ and $\mathscr{c} = L[j]$. Since $i \leq j$ then at the beginning of iteration i both $\mathscr{b}$ and $\mathscr{c}$ were in ns. Hence at iteration i, and by Lemma 3.2.14, $\mathscr{b} = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_i)$ and since $\mathscr{c} \in ns_i$ then $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{c}$, thus $L[i] \preccurlyeq_{\mathtt{LEX}} L[j]$.

We prove that if $L[i] \preccurlyeq_{\mathtt{LEX}} L[j]$ then $i \leq j$. Let $\mathscr{b} = L[i]$ and $\mathscr{c} = L[j]$. Suppose $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{c}$ and $i > j$. Then $\mathscr{c}$ was inserted into L before $\mathscr{b}$. At iteration j, and by Lemma 3.2.14, $\mathscr{c} = \mathsf{min}_{\preccurlyeq_{\mathtt{LEX}}}(ns_j)$, and since $\mathscr{b} \in ns_j$ then $\mathscr{c} \preccurlyeq_{\mathtt{LEX}} \mathscr{b}$.

Since $\mathscr{b} \preccurlyeq_{\mathtt{LEX}} \mathscr{c}$ and $\mathscr{c} \preccurlyeq_{\mathtt{LEX}} \mathscr{b}$ then $\mathscr{b} = \mathscr{c}$. Since an element cannot be inserted twice, $i = j$ which contradicts $i{>}j$.

Using Algorithm 2, we generate a total order denoted by $\preccurlyeq_{\mathtt{MSTP}}$, from $\sqsubseteq_{\mathtt{MSTP}}$ and $\preccurlyeq_{\mathtt{LEX}}$. After generating the total order over authorizations, we can now define the ch that applies the $\mathtt{MSTP}$ by choosing the smallest authorization w.r.t $\preccurlyeq_{\mathtt{MSTP}}$, formally, $\mathsf{ch} = \mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$.

Lemma 3.2.16 ($\mathtt{MSTP}$ strategy application) *Given a policy $P = (\mathscr{A}, \mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}})$, the function $\mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$ applies the $\mathtt{MSTP}$ strategy.*

Proof We prove that $\mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$ satisfies the $\mathtt{MSTP}$ property (see Definition 3.1.16). $\forall \mathscr{B} \subseteq \mathscr{A}$. By definition $\mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$ returns the smallest authorization with respect to $\mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$. Since $\mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$ is a linear extension of $\sqsubseteq_{\mathtt{MSTP}}$ then $\forall \mathscr{a}' \in \mathscr{B}.\ \mathscr{a}' \sqsubseteq_{\mathtt{MSTP}} \mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}} \Rightarrow \mathscr{a}' \sqsupseteq_{\mathtt{MSTP}} \mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}}$ hence the satisfaction of the $\mathtt{MSTP}$ property. ▮

Finally, the obtained structure $P = (\mathscr{A}, \mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}})$ is a fully-fledged policy.

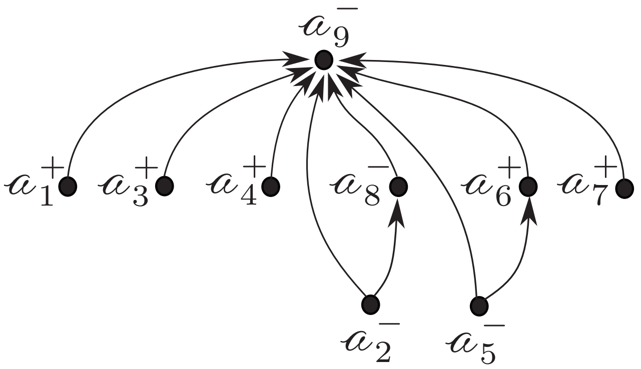


Figure 3.3: $\sqsubseteq_{\mathtt{MSTP}}$ DAG

Example 3.2.17 *Consider the policy $P = (\mathscr{A}, \mathsf{min}_{\preccurlyeq_{\mathtt{MSTP}}})$ where $\mathscr{A}$ is the set of authorizations $\mathscr{A}$ shown in Table 3.1. Algorithm 2 takes as parameters the total syntactical order of the Table 3.1, and the DAG $\mathsf{H}^{\mathscr{A}}_{\sqsubseteq_{\mathtt{MSTP}}}$ depicted by Figure 3.3. Table 3.3 shows the values of S and L at the end of each iteration of the loop between lines 4 and 15. The result of the Algorithm is a total order equal to $\preccurlyeq_{\mathtt{LEX}}$ since the latter is a linear extension of $\sqsubseteq_{\mathtt{MSTP}}$ as shown by Lemma 3.2.15.*

Table 3.3: S and L values at the end of each iteration

Iteration	S	L								
1	$\{a_3, a_4, a_2, a_5, a_7\}$	a_1								
2	$\{a_3, a_4, a_5, a_7, a_8\}$	a_1	a_2							
3	$\{a_4, a_5, a_7, a_8\}$	a_1	a_2	a_3						
4	$\{a_5, a_7, a_8\}$	a_1	a_2	a_3	a_4					
5	$\{a_7, a_8, a_6\}$	a_1	a_2	a_3	a_4	a_5				
6	$\{a_7, a_8\}$	a_1	a_2	a_3	a_4	a_5	a_6			
7	$\{a_8\}$	a_1	a_2	a_3	a_4	a_5	a_6	a_7		
8	$\{a_9\}$	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	
9	$\{\}$	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9

3.3 Conclusion

In this chapter, we introduced a fine-grained access control model for RDF stores with inference capabilities. 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. Instead of hard-coding the conflict resolution strategy, we proposed a more liberal approach, by defining an abstract conflict resolution function ch. The latter can be coded by the administrator during policy design time. We showed how concrete resolution strategies can be instances of our abstract framework. Moreover, we showed how more elaborate strategies, notably *most specific takes precedence*, can be built in our framework. To prevent a malicious user who knows the inference rules, from accessing confidential triples by applying these rules on his accessible triples, we propose in the next chapter a static verification approach to detect and fix the inference leakage problem.

Chapter 4

Inference leakage problem and solution

Contents

▷ *Access control models offer protection against direct accesses to sensitive information; however, indirect accesses to sensitive information may still be possible via inferences. Indeed, security violations via inferences called* inference problem *(inference leakage problem) occurs when sensitive information can be inferred from authorized data. In this chapter we show how static verification could be used to detect policy inconsistencies with respect to inference. In Section 4.1 we give an overview about the inference problem in different data models, and we show the different techniques to deal with it. In Section 4.2, we formally define the inference leakage problem which arises when confidential triples are inferred from authorized triples. We formalize the* consistency property *that captures the inference leakage problem. This property ensures that confidential information can not be inferred from authorized information with respect to a set of inference rules. To solve the issue, we propose in Section 4.3 a static verification algorithm which is run at policy design time. The algorithm checks if the interaction between policy authorizations and inference rules may lead to inference leakage. It generates a set of* counterexample *BGPs which represent graphs over which the policy presents an inference leakage problem. In Section 4.3.2 we show how to use the answer of the algorithm to fix the policy, or how to use them as integrity constraints that do not allow updates which could lead to inference leakage.* ◁

4.1 The inference problem

The inference problem (see [Farkas 2002] for a survey) exists in all types of database systems and has been studied extensively within the context of multilevel databases. Early works on the inference problem focused on statistical database security [Adam 1989]. The main requirement for securing statical databases is to allow users to make statistics by giving access to groups of entities while protecting the confidentiality of the individual entities. The problem is that a user might obtain confidential information about an individual entity by correlating different statistics. Another kind of inference problem appeared on data models which use semantic constraints such as functional dependencies [Su 1987]. Here, sensitive information can be disclosed from non-sensitive data and meta-data. We illustrate with the example used by [Su 1987]. Assume that a company database consists of the relation scheme EMP-SAL, which has three attributes: $Name$, $Rank$, and $Salary$. The relation $< Name, Salary >$ is a secret, but user u requests the following two queries: List $Rank$ and $Salary$ of all employees and List the $Name$ and $Rank$ of all employees. None of the queries violates the security requirement because they do not contain the secured $< Name, Salary >$ pair; however, suppose u is aware of the constraint that all employees having identical ranks have the same salaries, then she/he can infer the salary of employees.

Several works have been devoted to detect and deal with inference leakages. Inference control techniques can be divided according to the time of the inference detection control, *i.e.* static and dynamic approaches:

- **Static approaches** In the static approaches [Su 1987], all processing is done offline during design time. Once a security violation via inferences is detected, the system is modified to repair such violations. Modifications are done on the database schema or on the access control policy. The main advantage of this approach is that it is computationally less expensive than the dynamic approach. However, this approach may result in reduced availability of data because the inference problem may not materialize in a particular database instance.
- **Dynamic approaches** In the dynamic approaches ([Thuraisingham 1987]), the processing is done during runtime. Every issued query is checked whether or not it could lead to confidential data disclosure. If an inference violation is detected, the query is either refused or modified to avoid such violations. Most of these approaches keep a log of all queries issued by the user so they can be combined with the current one to check if the query is allowed or not. Whereas this kind of approaches increase availability, they are

computationally expensive, since the violation check is done for every query.

In the context of RDF, [Jain 2006] used a static approach by defining an algorithm that detects inference violations using the policy and the base graph (see Section 2.3). Since we focus on the publication of RDF triples in the context of linked data, we concentrate on fast query answering, hence we favor the static approach. Before we show how to detect whether a policy presents a problem of inference leakage, we define the consistency property.

4.2 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, inference 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. The interaction between authorizations and inference rules raises the issue of policy violation because it could lead to confidential data leakage. A malicious user who knows the inference rules could use a local reasoner and apply the inference rules over her/his accessible triples to infer triples she/he is not supposed to access. To illustrate this issue, consider the following example.

Example 4.2.1 *Assume a set of inference rules* $\mathbf{R} = \{\mathbf{RDom}, \mathbf{RAdm}\}$*, as shown in Example 2.1.16 (P. 22). We want to apply the policy defined in Example 3.1.10 (P. 50) on the graph* $\mathsf{Cl}_{\mathbf{R}}(G_0)$ *of Figure 2.2 (P. 16). According to Example 3.1.10 (P. 50), the authorized subgraph is* $(\mathsf{Cl}_{\mathbf{R}}(G_0))^+_P = \{et_1, et_4, et_5, et_6\}$*. 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 inference leakage. Figure 4.1 illustrates the inference leakage using* **RDom***.*

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

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

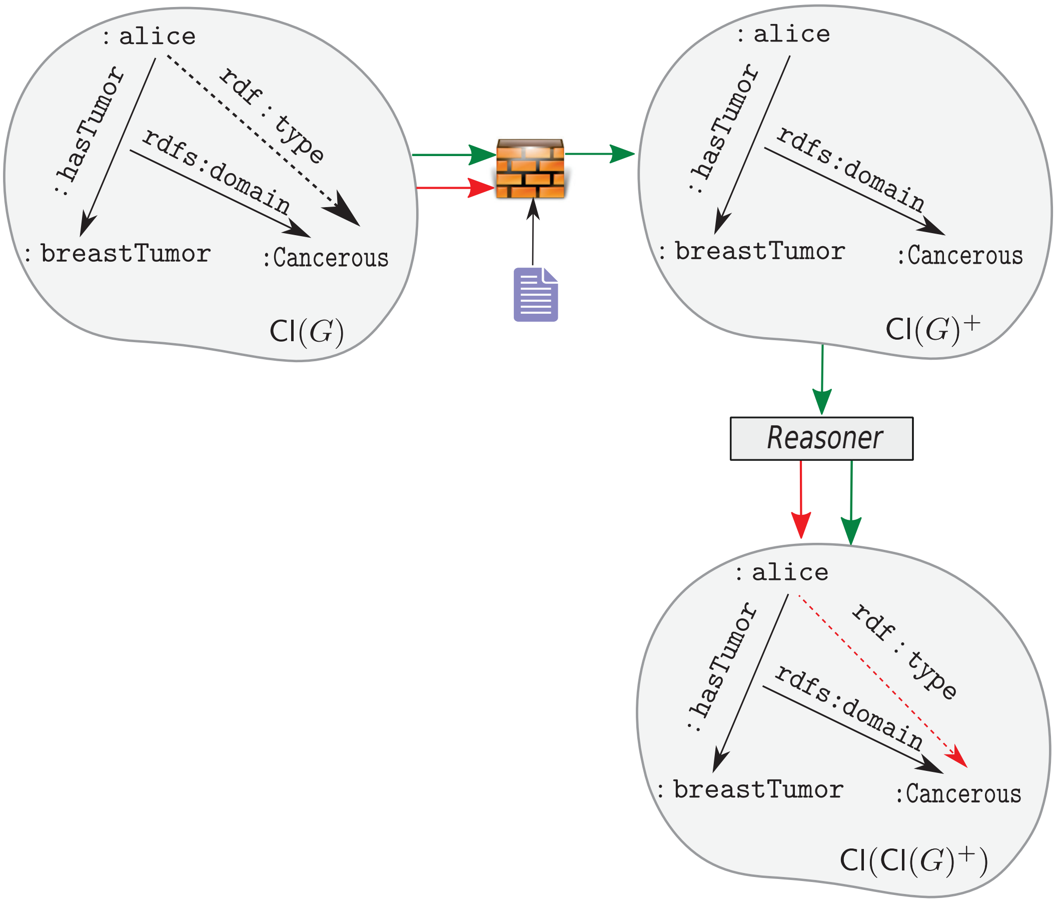


Figure 4.1: Inference leakage example

The consistency property has to hold for all graphs. It does not have to be checked when stored graphs are updated, but solely at the policy design-time or when the inference rules or the authorizations change.

4.3 Static verification

In computer programming, static code analysis provides a way to analyze the code in order to detect possible errors and security weaknesses [Livshits 2005]. In the context of access control, the static analysis could be used to check whether given properties are satisfied by the policies. 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. We define Algorithm 3 that, given an authorization policy $P = (\mathscr{A}, \mathsf{ch})$ and a set of inference rules $\mathbf{R}$, checks whether the consistency property (Definition 4.2.2) holds. Algorithm 3 is an enumeration algorithm and not a mere decision algorithm: it is constructive and finds all possible counterexamples to the consistency property.

Algorithm 3 Algorithm for enumerating inconsistency patterns

Input: a set of inference rules $\mathbf{R}$, an authorization policy $P = (\mathscr{A}, \mathsf{ch})$
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 $(\mathscr{a}_1, \ldots, \mathscr{a}_k, \mathscr{a}) \in \mathscr{A}^{+^k} \times \mathscr{A}^-$ do
            let $\rho_1, \ldots, \rho_k, \rho$ be renaming substitutions for $\mathscr{a}_1, \ldots, \mathscr{a}_k, \mathscr{a}$
            let $(ha_1, \ldots, ha_k, ha) = (\mathsf{head}(\mathscr{a}_1)\rho_1, \ldots, \mathsf{head}(\mathscr{a}_k)\rho_k, \mathsf{head}(\mathscr{a})\rho)$
            if $\exists \mu = \mathsf{mgu}((ha_1, \ldots, ha_k, ha), (tp_1, \ldots, tp_k, tp))$ then
                let $B = \bigcup_{i=1}^{k} \mathsf{hb}(\mathscr{a}_i)\rho_i\mu \cup \mathsf{hb}(\mathscr{a})\rho\mu$
                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
```

The principle of Algorithm 3 is to find an inference rule $(tp \leftarrow tp_1, \ldots, tp_k) \in \mathbf{R}$ and related sets of authorizations $(\mathscr{a}_1, \ldots, \mathscr{a}_k, \mathscr{a})$ such that $\mathscr{a}$ is negative and its head is unifiable with tp and all authorizations $\mathscr{a}_i$ for $i \in \{1, \ldots, k\}$ are positive and their heads are unifiable with $\{tp_1, \ldots, tp_k\}$. Pictorially:

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

Intuitively, let us consider the graph B built by considering the union of the underlying graphs $\mathsf{hb}(\mathscr{a}_1) \ldots \mathsf{hb}(\mathscr{a}_k)$ and $\mathsf{hb}(\mathscr{a})$, properly renamed and unified. By construction, the inference rule $\mathbf{r}$ is applicable. 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 $\mathscr{a}$: an inconsistency.

The key idea that ensures the completeness of Algorithm 3 is that all counterexamples of the consistency property have to arise this way. Theorems 4.3.1 and 4.3.2 formally state the correctness of the algorithm: P is not consistent w.r.t. $\mathbf{R}$ *if and only if* Algorithm 3 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, [Martelli 1982]. (See Appendix A, Section A.2).

Theorem 4.3.1 (Soundness of Algorithm 3) *If Algorithm 3 returns a non empty collection then P is not consistent w.r.t. $\mathbf{R}$.*

Proof Let $B \in \textsc{RdfLeaks}(\mathbf{R}, P)$. Since $\{(tp_1)\mu, \ldots, (tp_k)\mu\} \subseteq (\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}$, we have $\mu \in [\![\{tp_1, \ldots, tp_k\}]\!]_{(\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}}$. Thus $(tp)\mu \in \phi_r((\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}) \subseteq \mathsf{Cl}_{\mathbf{R}}((\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P})$. However $(tp)\mu \notin (\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}$. Therefore $\mathsf{Cl}_{\mathbf{R}}((\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}) \neq (\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}$, thus P is not consistent w.r.t. $\mathbf{R}$. ∎

Theorem 4.3.2 (Completeness of Algorithm 3) *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 $B \in \textsc{RdfLeaks}(\mathbf{R}, P)$ such that $[\![B]\!]_{\mathsf{Cl}_{\mathbf{R}}(G)} \neq \emptyset$.*

Theorem 4.3.1 holds by construction: Line 9 of Algorithm 3 ensures that B is a counterexample. Next, we prove Theorem 4.3.2 and discuss counterexample usage.

4.3.1 Proof of completeness

To show that Theorem 4.3.2 holds, we first introduce two lemmas. Intuitively, Lemma 4.3.3 ensures that the Definition 3.1.3 (P. 47) of applicable authorizations behaves well according to graphs inclusion. Lemma 4.3.4 is its counterpart for the closure of a graph according to a set of inference rules.

Lemma 4.3.3 *Let $P = (\mathscr{A}, \mathsf{ch})$ 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 $t \in B$, $\mathsf{ar}(B, \mathscr{A})(t) \subseteq \mathsf{ar}(G, \mathscr{A})((t)\eta)$.*

Proof Let $\mathscr{a}$ be any authorization in $\mathsf{ar}(B, \mathscr{A})(t)$. By definition 3.1.3 (P. 47), there exists θ such that $t = \mathsf{head}(\mathscr{a})\theta$ and $\mathsf{hb}(\mathscr{a})\theta \subseteq B$. Thus $t\eta = \mathsf{head}(\mathscr{a})\theta\eta$ and $\mathsf{hb}(\mathscr{a})\theta\eta \subseteq B\eta \subseteq G$. Thus, by definition 3.1.3 (P. 47), $\mathscr{a} \in \mathsf{ar}(G, \mathscr{A})((t)\eta)$.
∎

Lemma 4.3.4 *Let $P = (\mathscr{A}, \mathsf{ch})$ 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 First, we prove that for any $B', G' \in \mathsf{BGP}$, if $B'\eta \subseteq G'$ and if $t \in \phi_{\mathbf{R}}(B')$, then $(t)\eta \in \phi_R(G')$. By definition 2.1.13 (P. 21) there exists a rule $\mathbf{r} = (tp \leftarrow tp_1, \ldots, tp_k) \in \mathbf{R}$ and a substitution σ such that $\{(tp_1)\sigma, \ldots, (tp_k)\sigma\} \subseteq B'$ and $t = (tp)\sigma$. Thus $\{(tp_1)\sigma\eta, \ldots, (tp_k)\sigma\eta\} \subseteq (B')\eta \subseteq G'$ and $t = (tp)\sigma\eta$. Thus, by definition 2.1.13 (P. 21), $(t)\eta = (tp)\sigma\eta \in \phi_{\mathbf{R}}(G')$.

Let $(B_i)_{i\in\mathbb{N}}$ (resp. $(G_i)_{i\in\mathbb{N}}$) be the sequence defining $\mathsf{Cl}_{\mathbf{R}}(B)$ (resp. $\mathsf{Cl}_{\mathbf{R}}(G)$) as in definition 2.1.13 (P. 21). We prove, by induction on i, that for all $i \in \mathbb{N}$ if $t \in B_{i+1}$, then $(t)\eta \in G_{i+1}$ (*i.e.* $(B_{i+1})\eta \subseteq G_{i+1}$). If $t \in B$, then $(t)\eta \in (B)\eta \subseteq G \subseteq G_{i+1}$. If $t \in B_i$, then, by induction hypothesis, $(t)\eta \in G_i \subseteq G_{i+1}$. Otherwise $t \in \phi_{\mathbf{R}}(B_i)$. As previously proved and since, by induction hypothesis, $(B_i)\eta \subseteq G_i$, this means that $(t)\eta \in \phi_{\mathbf{R}}(G_i) \subseteq G_{i+1}$.

Since for any $i \in \mathbb{N}$, $(B_i)\eta \subseteq G_i$, we conclude that $(\mathsf{Cl}_{\mathbf{R}}(B))\eta = \bigcup_{i\in\mathbb{N}}(B_i)\eta \subseteq \bigcup_{i\in\mathbb{N}} G_i = \mathsf{Cl}_{\mathbf{R}}(G)$. ∎

Lemma 4.3.5 *Let $P = (\mathscr{A}, \mathsf{ch})$ be a policy, $\mathscr{a} \in \mathscr{A}$ be an authorization, G a basic graph pattern and $t \in G$. If $\mathscr{a} \in \mathsf{ar}(G, \mathscr{A})(t)$, then for any renaming substitution ρ of $\mathscr{a}$, there exists a substitution θ such that $(\mathsf{head}(\mathscr{a})\rho)\theta = t$ and $\theta \in [\![\mathsf{hb}(\mathscr{a})\rho]\!]_G$.*

Proof By definition 3.1.3 (P. 47), there exists a substitution θ' such that $t = (\mathsf{head}(\mathscr{a}))\theta'$ and $\theta' \in [\![\mathsf{hb}(\mathscr{a})]\!]_G$, *i.e.* $\mathsf{hb}(\mathscr{a})\theta' \in G$. Since ρ is a renaming substitution, it is bijective and $(\mathsf{hb}(\mathscr{a})\rho)\rho^{-1} = \mathsf{hb}(\mathscr{a})$ and $(\mathsf{head}(\mathscr{a})\rho)\rho^{-1} = \mathsf{head}(\mathscr{a})$.

Thus $((\mathsf{hb}(\mathscr{a})\rho)\rho^{-1})\theta' \in G$ and $t = ((\mathsf{head}(\mathscr{a})\rho)\rho^{-1})\theta'$. It is sufficient to take $\theta = \rho^{-1}\theta'$. ∎

Lemma 4.3.6 *Let $P = (\mathscr{A}, \mathsf{ch})$ be an authorization policy, $\mathbf{R}$ a set of inferences rules, $B, G \in \mathsf{BGP}$ basic graph patterns, and η a substitution such that $B\eta \subseteq G$. For any $t \in B$, $\mathsf{ar}(\mathsf{Cl}_R(B), \mathscr{A})(t) \subseteq \mathsf{ar}(\mathsf{Cl}_R(G), \mathscr{A})((t)\eta)$.*

Proof By Lemma 4.3.4, $(\mathsf{Cl}_R(B))\eta \subseteq \mathsf{Cl}_R(G)$. By applying Lemma 4.3.3 on $\mathsf{Cl}_R(B)$ and $\mathsf{Cl}_R(G)$, we conclude that for any $t \in B \subseteq \mathsf{Cl}_R(B)$, $\mathsf{ar}(\mathsf{Cl}_R(B), \mathscr{A})(t) \subseteq \mathsf{ar}(\mathsf{Cl}_R(G), \mathscr{A})((t)\eta)$. ∎

Lemma 4.3.7 *If $\mathsf{Cl}_{\mathbf{R}}(G) \neq G$, then there exists $t \in \phi_{\mathbf{R}}(G) \subseteq \mathsf{Cl}_{\mathbf{R}}(G)$ such that $t \notin G$.*

Proof By definition 2.1.13, $\mathsf{Cl}_{\mathbf{R}}(G) = \bigcup_{i \in \mathbb{N}} G_i$, where $G_0 = G$ and $G_{i+1} = \phi_{\mathbf{R}}(G)$. By induction on i, we prove that if $G \neq G_{i+1}$ then there exists $t \in \phi_{\mathbf{R}}(G)$ such that $t \notin G$. If $G \neq G_{i+1}$, and since $G \subseteq G_{i+1}$, there exists $t' \in G_{i+1}$ such that $t' \notin G$. If $t' \in G_i$, then $G_i \neq G$ and we get the result by induction hypothesis on G_i. If $t' \notin G_i$, then $t' \in \phi_{\mathbf{R}}(G_i)$. If $G_i = G$, then it is sufficient to take $t = t'$. Otherwise $G_i \neq G$ and we get the result by induction hypothesis on G_i. ∎

Proof of Theorem 4.3.2 Let G^{ex} be such that $\mathsf{Cl}_{\mathbf{R}}((\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}) \neq (\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}$. By Lemma 4.3.7, there exists $t^{ex} \in \phi_{R}((\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P})$ such that $t^{ex} \notin (\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}$.

By Definition 2.1.13 (P. 21), there is a rule $\mathbf{r} = (tp \leftarrow tp_1, \ldots, tp_k) \in \mathbf{R}$ and a substitution σ such that $t^{ex} = (tp)\sigma$ and $\sigma \in [\![\{tp_1, \ldots, tp_k\}]\!]_{(\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}}$.

Let us consider $i \in \{1, \ldots, k\}$. Since $\sigma \in [\![\{tp_1, \ldots, tp_k\}]\!]_{(\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}}$, it is the case that $(tp_i)\sigma \in (\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}$. Let $\mathscr{a}_i = \mathsf{ch}(\mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(G^{ex}), \mathscr{A})((tp_i)\sigma))$ be the authorization selected for $(tp_i)\sigma$ in $\mathsf{Cl}_{\mathbf{R}}(G^{ex})$. Since $(tp_i)\sigma \in (\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}$, $\mathscr{a}_i \in \mathscr{A}^{+}$. Similarly and since $(tp)\sigma = t^{ex} \notin (\mathsf{Cl}_{\mathbf{R}}(G^{ex}))^{+}_{P}$, if we pose $\mathscr{a} = \mathsf{ch}(\mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(G^{ex}), \mathscr{A})((tp)\sigma))$ then $\mathscr{a} \in \mathscr{A}^{-}$. Thus $(\mathscr{a}_1, \ldots, \mathscr{a}_k, \mathscr{a}) \in {\mathscr{A}^{+}}^{k} \times \mathscr{A}^{-}$, and Algorithm 3 evaluates this combination of rule and authorizations at Line 4.

Let us assume ρ (resp. for any $i \in \{1, \ldots, k\}$, ρ_i), ha (resp. ha_i) as they are defined by Algorithm 3 at Lines 5 and 6. Since P respects the *Closedness* and the *Totality* conditions of Definition 3.1.7 (P. 49), $\mathscr{a} \in \mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(G^{ex}), \mathscr{A})((tp)\sigma)$ (resp. for any $i \in \{1, \ldots, k\}$, $\mathscr{a}_i \in \mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(G^{ex}), \mathscr{A})((tp_i)\sigma)$). That is, by Lemma 4.3.5, there exists a substitution θ (resp. θ_i) such that:

$$
\begin{aligned}
(tp)\sigma &= (ha)\theta && (1)\\
(tp_i)\sigma &= (ha_i)\theta_i && (2)\\
\theta &\in [\![\mathsf{hb}(\mathscr{a})\rho]\!]_{\mathsf{Cl}_{\mathbf{R}}(G^{ex})} && (3)\\
\theta_i &\in [\![\mathsf{hb}(\mathscr{a}_i)\rho_i]\!]_{\mathsf{Cl}_{\mathbf{R}}(G^{ex})} && (4)
\end{aligned}
$$

Because $\rho, \rho_1, \ldots, \rho_k$ are renaming substitutions, $\mathsf{var}(ha), \mathsf{var}(ha_1), \ldots, \mathsf{var}(ha_k)$ and $\mathsf{var}(\{tp, tp_1, \ldots, tp_k\})$ are pairwise disjoint. Thus we can introduce the following substitution μ':

$$\mu'(x) = \begin{cases} \theta(x) & \text{if } x \in \mathsf{var}(ha) \\ \theta_i(x) & \text{if } x \in \mathsf{var}(ha_i) \\ \sigma(x) & \text{if } x \in \mathsf{var}(\{tp, tp_1, \ldots, tp_k\}) \end{cases}$$

One can remark that $(ha)\mu' = (tp)\mu'$ (resp. for any $i \in \{1, \ldots, k\}$, $(ha_i)\mu' = (tp_i)\mu'$). Therefore, there exists $\mu = \mathsf{mgu}((ha_1, \ldots, ha_k, ha), (tp_1, \ldots, tp_k, tp))$, and there exists a substitution η such that $\mu' = \mu\eta$ so condition at Line 7 is satisfied.

Let us consider $B = \bigcup_{i=1}^{k} \mathsf{hb}(\mathpzc{a}_i)\rho_i\mu \cup \mathsf{hb}(\mathpzc{a})\rho\mu$ as in Algorithm 3 at Line 8. By definition of μ' and because of (1)-(4), $B\eta = (\bigcup_{i=1}^{k} \mathsf{hb}(\mathpzc{a}_i)\rho_i\mu' \cup \mathsf{hb}(\mathpzc{a})\rho\mu') \subseteq \mathsf{Cl}_{\mathbf{R}}(G^{ex})$.

Since $\mathsf{hb}(\mathpzc{a})\rho\mu \subseteq B$ and $\mathsf{head}(\mathpzc{a})\rho\mu = (ha)\mu$, we have $\mathpzc{a} \in \mathsf{ar}(B, \mathscr{A})((ha)\mu)$. One can remark that $(tp)\sigma = (tp)\mu' = (ha)\mu' = ((ha)\mu)\eta$, $(tp)\sigma \in \mathsf{Cl}_{\mathbf{R}}(G^{ex})$ and $B\eta \subseteq \mathsf{Cl}_{\mathbf{R}}(G^{ex})$. Thus by Lemma 4.3.6, $\mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(B), \mathscr{A})((ha)\mu) \subseteq \mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(\mathsf{Cl}_{\mathbf{R}}(G^{ex})), \mathscr{A})((tp)\sigma) = \mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(G^{ex}), \mathscr{A})((tp)\sigma)$, as $\mathsf{Cl}_{\mathbf{R}}(\mathsf{Cl}_{\mathbf{R}}(G^{ex})) = \mathsf{Cl}_{\mathbf{R}}(G^{ex})$. Moreover, since $\mathpzc{a} \in \mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(B), \mathscr{A})((ha)\mu)$ and $\mathpzc{a} = \mathsf{ch}(\mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(G^{ex}), \mathscr{A})((tp)\sigma))$, we deduce by the *Monotony* condition of Definition 3.1.7 (P. 49) applied on P that $\mathpzc{a} = \mathsf{ch}(\mathsf{ar}(\mathsf{Cl}_{\mathbf{R}}(B), \mathscr{A})((ha)\mu))$. Since $\mathsf{effect}(\mathpzc{a}) = -$, we conclude that $(tp)\mu = (ha)\mu \notin (\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}$.

Similarly, for any $i \in \{1, \ldots, k\}$, we deduce that $(tp_i)\mu \in (\mathsf{Cl}_{\mathbf{R}}(B))^{+}_{P}$. Therefore B is added in $BGPs$ by Algorithm 3 at Line 10. Thus there is $B \in \textsc{RdfLeaks}(\mathbf{R}, P)$ and η such that $\eta \in [\![B]\!]_{\mathsf{Cl}_{\mathbf{R}}(G^{ex})}$. ∎

4.3.2 Understanding the Counterexamples

As Algorithm 3 enumerates inconsistency patterns, its output can be used to correct the access control policy. A proof of concept of the algorithm has been implemented in Prolog[1]. The methodology to correct an inconsistent policy, as shown in Figure 4.2, is to iteratively apply the following two steps: (1) use Algorithm 3 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, in other words when Algorithm 3 returns an empty set.

[1] http://liris.cnrs.fr/~tsayah/AC4RDF/

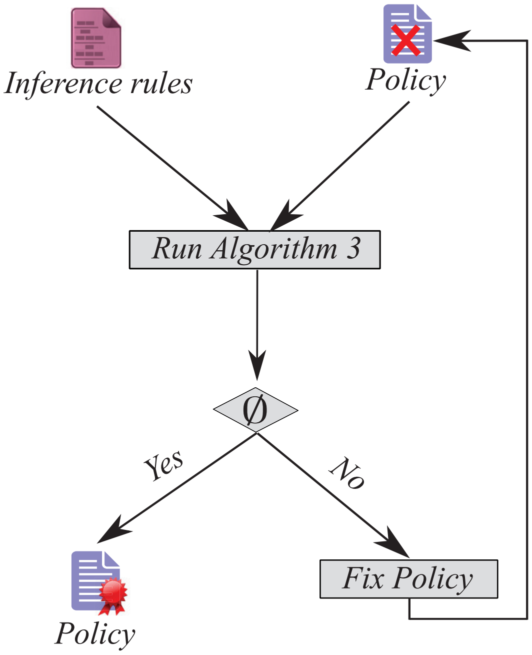
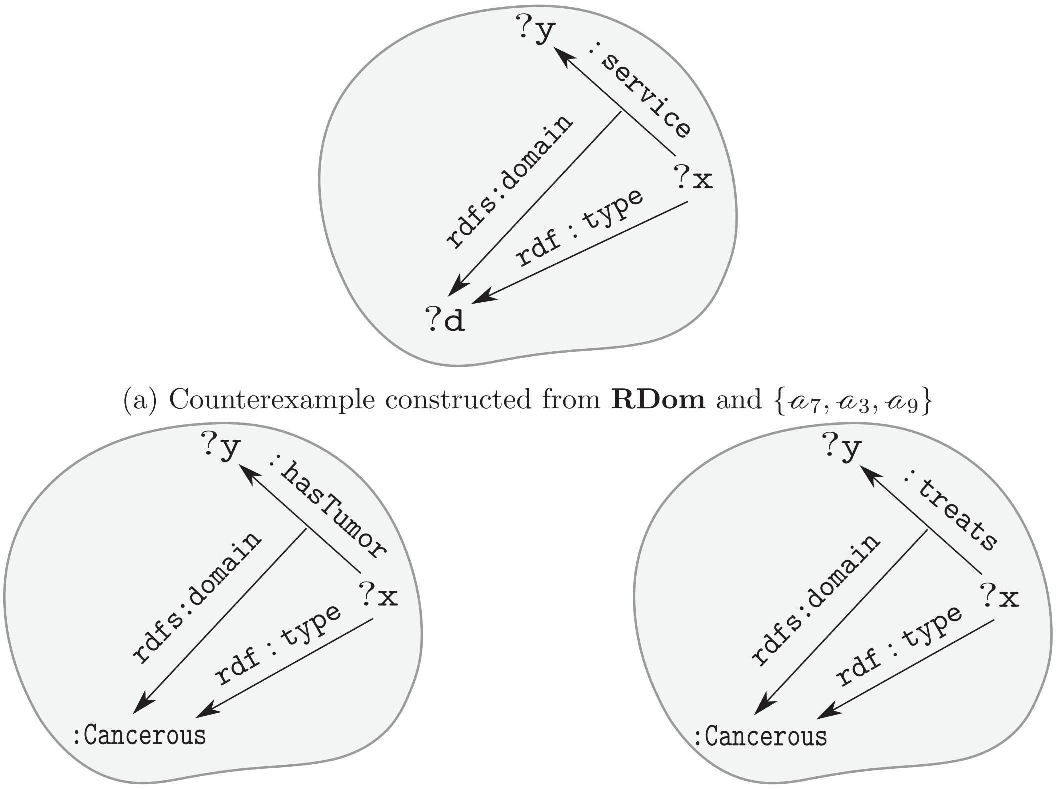


Figure 4.2: Iterative Policy fix

We illustrate this methodology on the inference rules of Example 2.1.16 (P. 22) and the policy defined in Table 3.1 (P. 47) with syntactical order. After three iterations, no inconsistency subsists anymore. The complete policy once corrected is given in Table 4.1.

The first run points out the fact that triples with predicate `rdf :type` are forbidden by the default authorization a_9, but they can be deduced by rule **RDom**. Figure 4.3a depicts a counterexample generated from **RDom**, positive authorizations a_7, a_1 and the negative universal authorization a_9. Since most of `rdf :type` predicates information should be public for integration purposes, we add a new authorization a'_8 that grants access to triples with predicate `rdf :type`. a'_8 is added after a_8 so that the triples with predicate `rdf :type` will be granted except those which object is `:Cancerous`.

A second run points out that triples of the form (`X`; `rdf :type`; `:Cancerous`) can be deduced by rule **RDom** since triples (`Y` ; `rdfs :domain` ; `:Cancerous`) are accessible. Figure 4.3b depicts a counterexample generated from **RDom**, positive authorizations a_7, a_1 and negative authorization a_2, *i.e.* $B = \{$(`?x` ; `:hasTumor` ; `?y`), (`:hasTumor` ; `rdfs :domain` ; `:Cancerous`), (`?x` ; `rdf :type` ; `:Cancerous`)$\}$. Note that the triple (`:hasTumor` ; `rdfs :domain` ; `:Cancerous`) has been granted because its applicable authorizations are a_7 and a_8, and `ch` chose the first applicable one which is the positive authorization a_7. To remove



(a) Counterexample constructed from **RDom** and $\{\mathscr{a}_7, \mathscr{a}_3, \mathscr{a}_9\}$

(b) Counterexample constructed from **RDom** and $\{\mathscr{a}_7, \mathscr{a}_1, \mathscr{a}_2\}$

(c) Counterexample constructed from **RDom** and $\{\mathscr{a}_7, \mathscr{a}_4, \mathscr{a}_2\}$

Figure 4.3: Counterexamples constructed from **RDom**

such source of inconsistencies, one can switch the order of $\mathscr{a}_7$ and $\mathscr{a}_8$, denying access to (`Y` ; `rdfs :domain` ; `:Cancerous`). After switching the two authorizations, at the next run, the Algorithm constructs the same BGPs but it does not insert them into the set of counterexamples. For instance, the BGP B of Figure 4.3b will be constructed at line 8 but the test at line 9 will not succeed since (`:hasTumor` ; `rdfs :domain` ; `:Cancerous`) is denied by $\mathscr{a}_8$, thus it does not belong to $(\mathsf{Cl}_{\mathbf{R}}(B))^+_P$.

Note that even the counterexamples generated by the Algorithm 3 are valid RDF graphs, they may be inconsistent w.r.t a given vocabulary. For instance, the counterexample depicted by Figure 4.3c, states that `:treats` domain is `:Cancerous`, whereas in our RDFS vocabulary the triple et_1 in Figure 2.4 states that it is the domain of `:hasTumor`. As handling vocabularies is out of the scope of this thesis, we will discuss this issue and possible solutions in Chapter 7.

We give more details about the third run that produces a single counterexample graph depicted by Figure 4.4, *i.e.* $B = \{($`?d` ; `:service` ; `?s`),

Table 4.1: Corrected authorization policy

a_1	= GRANT(?p ; :hasTumor ; ?t)
a_2	= DENY (?p ; rdf :type ; :Cancerous)
a_3	= GRANT(?d ; :service ; ?s)
a'_3	= DENY (?d ; :treats ; ?p) WHERE {(?d ; :service ; ?s), (?s ; rdf :type ; :Oncology)}
a_4	= GRANT(?d ; :treats ; ?p)
a_5	= DENY (?p ; :admitted ; ?s) WHERE {(?s ; rdf :type ; :Oncology)}
a_6	= GRANT(?p ; :admitted ; ?s)
a_8	= DENY (?s ; ?p ; :Cancerous)
a_7	= GRANT(?p ; rdfs :domain ; ?s)
a'_8	= GRANT (?x0 ; rdf :type ; ?x1)
a_9	= DENY (?s ; ?p ; ?o)

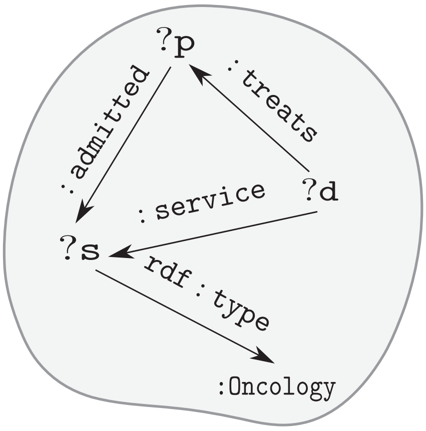


Figure 4.4: Counterexample constructed from **RAdm** and $\{a_3, a_4, a_5\}$

(?d ; :treats ; ?p), (?p ; :admitted ; ?s), (?s ; rdf :type ; :oncology)} which involves the rule **RAdm** together with authorizations a_3, a_4 and a_5. A first and simple solution would be to change the effect of authorization 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 a'_3 just before a_4. a'_3 denies access to triples of doctors treating patients only if they work in an oncology service. Note that we can alternatively switch (?d ; :treats ; ?p) and (?d ; :service ; ?s) in a'_3, but such a choice should be discussed with domain experts first. After adding a'_3, a final execution of the algorithm confirms that the new policy is consistent w.r.t $\{\mathbf{RDom}, \mathbf{RAdm}\}$ as it returns no counterexample.

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 4.3.2, if there is no such instance, no inference leakage will occur. Thus, one could use each B produced by Algorithm 3 as an integrity constraint in the RDF store, thereby reject updates that may lead to inference leakage.

In order to guarantee the absence of inference leakage, one has to add such an integrity constraint for every graph pattern in the output of Algorithm 3.

Consider the aforementioned counterexample of Figure 4.4, this means that no leakage corresponding to $\mathscr{a}_5$ and resulting from $\mathscr{a}_3, \mathscr{a}_4$ and **RDom** will occur. However, in case of a high number of counterexamples, this approach may burden the query evaluation.

4.4 Conclusion

In this chapter we formalized an inference 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. Note that OWL2RL, an OWL profile, can also be statically verified since it is described as a collection of positive Datalog rules [Kolovski 2010]. We proposed a correct and complete algorithm which generates a set of counterexample patterns in case the policy presents an inference leakage problem. We showed how these counterexamples could be used by the administrator to fix the policy or as integrity constraints of the base graph. We did some experiments with the implemented Prolog program, using RDFS rules and a set of authorizations. The experiments showed that some RDFS inference rules which have only variables in the premises, cause the generation of several counterexamples, since the premises are unifiable with any authorization's head. One of these rules is : $\frac{(\texttt{?x ;?y ;?z})}{(\texttt{?x ;rdf:type ;rdfs:Resource})}$. It caused the generation of many counterexamples since in the set of our authorizations, we did not grant access to triples which object is `rdfs :Resource`, and they were denied by default by the universal authorization. Since this kind of triples does not need to be private, adding a grant authorization that gives access to them, reduced considerably the number of counterexamples.

Chapter 5

Policy administration

Contents

▷ *In Chapter 3 we defined `AC4RDF`, a fine grained access control model for RDF which semantics are defined by means of positive subgraph from the base graph. The `AC4RDF` policies are defined for a given requester and we mentioned that the upstream mapping from requesters to authorizations may use any model from the literature. In this chapter we propose a high level access control language which allows the definition of high level global policies which are then compiled to `AC4RDF` policies to be enforced. We choose to define our policies on the basis of the user's attributes following the Attribute Based Access Control (ABAC) which allows a much more fine-grained access control approach combining not only user attributes but other environment information, such as location and time. First we define the main components of the proposed solution then we give the syntax and semantics of a XACML-like policy language. Policies are defined by trees where nodes represent policies, the leaves represent authorizations and edges represent targets. A target defines certain conditions to determine whether this policy is applicable to a user's request. Finally, we show how the user's policy is generated and enforced.* ◁

[Costabello 2012] proposed an attribute based access control at named graph level. The intercepted user query is rewritten with respect to the user's attributes by adding the accessible named graphs to the query which is then evaluated on the dataset. Similarly, [Abel 2007] proposed a query rewriting technique. The user attributes are first bound boolean expressions. If it is evaluated to true, then the query is expanded using policies and evaluated over the dataset. The drawback of this kind of approaches is that they depend on the query language. [Reddivari 2005] proposed a language in which policies are based on user's attributes that are modeled as RDF triples and stored in the RDF store. They gave examples on static attributes such as *allow A to see the salary of B if A is the supervisor of B*. They did not mention how to treat dynamic attributes such as time and position. [Lopes 2012] proposed a model where triples are annotated with *non-recursive Datalog with negation* programs which evaluation determines the access permission to the triple given a specific set of attributes.

In this chapter we give the syntax and the semantics of the language that allows the definition of global policies. We define a XACML-like language which allows to describe global policies. Our intuition is to represent the policy in *Tree structure* where the intermediate nodes represent policies, the leaves are authorizations and nodes edges are labeled with *targets* representing attribute based conditions. To illustrate, consider the following example.

Example 5.0.1 *Suppose we want to protect RDF data about patients and their medical records. The hospital policy rules state that:*

- R_1 : *The administration staff can access all information about patients during working time only.*
- R_2 : *The administration staff can access all information in the medical records during working time, except the patient's disease.*
- R_3 : *Doctors and nurses can access medical records except those belonging to patients admitted to oncology.*
- R_4 : *Doctors working in oncology service can access medical records of patients admitted to oncology service.*
- R_5 : *Doctors and nurses can access medical records belonging to patients admitted to oncology only if the patient is in critical condition.*

First we translate the objects and conditions of the above rules by means of authorizations and targets. R_1 *is translated by one authorization* $\mathscr{a}_4$ *which*

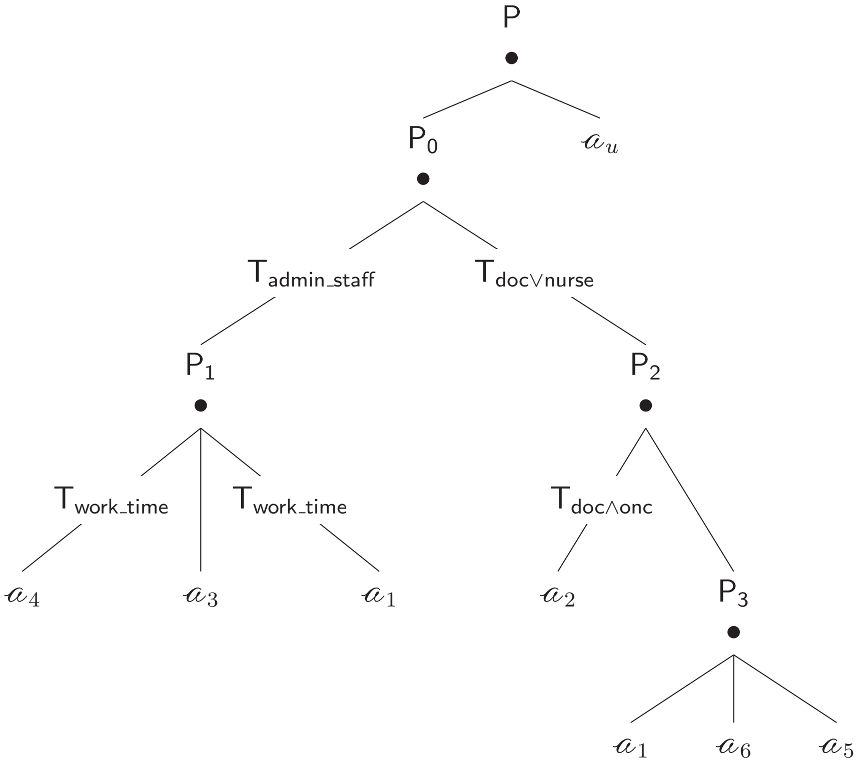

P
P_0
a_u
$\mathsf{T}_{\mathsf{admin\_staff}}$
$\mathsf{T}_{\mathsf{doc} \vee \mathsf{nurse}}$
P_1
P_2
$\mathsf{T}_{\mathsf{work\_time}}$
$\mathsf{T}_{\mathsf{work\_time}}$
$\mathsf{T}_{\mathsf{doc} \wedge \mathsf{onc}}$
a_4
a_3
a_1
a_2
P_3
a_1
a_6
a_5

Figure 5.1: Policy Tree

grants access to all the patient's information. R_1 has two conditions, the user must belong to the administration staff which is translated by target $\mathsf{T}_{\mathsf{admin\_staff}}$ and the access must be at working time, translated by target $\mathsf{T}_{\mathsf{work\_time}}$. R_2 is translated by two authorizations a_1 and a_3. a_1 grants access to all information in medical records. a_3 is an exception of a_1 since it denies access to disease name in the medical record. R_2 has the same conditions as R_1. R_3 is translated by authorizations a_1 and a_5. a_5 is an exception of a_1 that denies access to medical records of patients admitted to oncology. R_3 condition is that the user must be a doctor or a nurse, translated by target $\mathsf{T}_{\mathsf{doc} \vee \mathsf{nurse}}$. R_4 is translated by authorization a_2 which grants access to medical records of patients admitted to oncology. R_4 condition is that the user is a doctor working in oncology, translated by target $\mathsf{T}_{\mathsf{doc} \wedge \mathsf{onc}}$. R_5 is translated by authorization a_6 which is an exception a_5 that grants access to medical records of patients admitted to oncology and who are in critical condition. R_5 has the same conditions as R_3. Figure 5.1 depicts the above policy in a tree data structure.

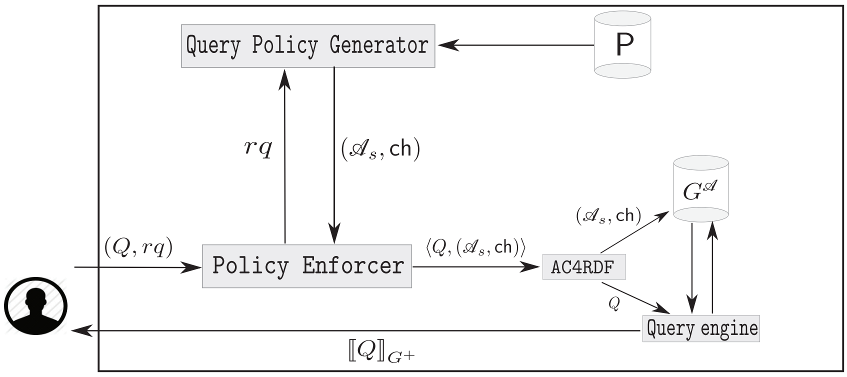

Query Policy Generator
P
rq
$(\mathscr{A}_s, \mathsf{ch})$
$G^{\mathscr{A}}$
$(\mathscr{A}_s, \mathsf{ch})$
(Q, rq)
Policy Enforcer
$\langle Q, (\mathscr{A}_s, \mathsf{ch})\rangle$
AC4RDF
Q
Query engine
$\llbracket Q \rrbracket_{G^+}$

Figure 5.2: System architecture

5.1 System architecture

As shown in Figure 5.2, our solution has three main components:

- `Policy Enforcer` intercepts the user query Q and attributes rq and returns the result $\llbracket Q \rrbracket_{G^+}$ to the user.
- `User Policy Generator` is responsible for the constitution of the user policy $(\mathscr{A}_s, \mathsf{ch})$ using the user's attributes sent from the `Policy Enforcer`.
- `AC4RDF` receives the user's policy and her/his query from the `Policy Enforcer` and answers with the result of the query over the user's positive subgraph.

The `Policy Enforcer` is a simple component which plays the role of interconnection between the user, the `User Policy Generator` and `AC4RDF`. `AC4RDF` was defined in Chapter 3 where we showed how the user positive subgraph is computed from the base graph using a user policy. The `User Policy Generator` allows to construct a user policy using her/his attributes and a *Global Policy*.

5.2 Language syntax and semantics

To formally define the semantics of our language and reasoning on it, we provide a syntax of our formalism through the Extended BNF (EBNF) grammar in Table 5.1. "*" (The Kleene Star) means 0 or more occurrences, whereas "+" (The Kleene Cross) means 1 or more occurrences. Note that Auth, OP, Key and Value are pairwise disjoint.

Table 5.1: Syntax of the policy language

GPolicy :=	(Target; GPolicy)$^+$
|	Auth
Target :=	(Target $\wedge$ Target)
|	(Target $\vee$ Target)
|	($\neg$Target)
|	Atom
|	$\top$
Atom :=	Key OP Value
|	Key OP Key
OP :=	$\underset{\cdot}{<} \mid \underset{\cdot}{>} \mid \underset{\cdot}{\leq} \mid \underset{\cdot}{\geq} \mid \underset{\cdot}{=} \mid \ldots$
Key :=	$\mathtt{k}_0 \mid \ldots \mid \mathtt{k}_n$
Value :=	$\mathtt{v}_0 \mid \ldots \mid \mathtt{v}_m$
Auth :=	$\mathscr{a}_0 \mid \ldots \mid \mathscr{a}_q \mid \mathscr{a}_u$
Request :=	(Key; Value)*

5.2.1 Abstracting Policy Components

The global policy is made of multiple components which are defined as follows:

Request is the component used to query the GPolicy to get the necessary information about policies that are applicable to the user. The latter represents the entity requesting to perform an operation upon the policy objects, and is characterized by attributes. In our language we define attributes as a pair (Key, Value) where Key specifies the attribute's name such as user-id, date of birth, home address, job function . . . , and Value specifies the value. A user request is defined by a list of pairs (Key, Value). The evaluation of a request over a GPolicy amounts to compare the Key and the Value according to the criteria defined in GPolicy, more precisely in the Target. In other words the Request component represents meta-data of the user, and not the action she/he wants to execute. Note that the attributes authenticity is out of the scope of this thesis. Several mechanisms exist to handle attributes authenticity such as *attribute certificate* [Farrell 2002].

Example 5.2.1 *The following illustrates examples of requests:*

$$
\begin{aligned}
\mathsf{rq}_0 &= [(role, \mathsf{doctor}), (role, \mathsf{admin\_staff}), (service, \mathsf{onc})] \\
\mathsf{rq}_1 &= [(role, \mathsf{admin\_stuff}), (time, \text{"}\mathsf{09:00}\text{"})] \\
\mathsf{rq}_2 &= [(role, \mathsf{nurse})]
\end{aligned}
$$

GPolicy is the main component of our formalism and it is the root of all policy components. A GPolicy can be an atomic authorization or a list of pairs of targets and *child policies* which themselves belong to GPolicy.

Auth is the atomic component which represents the policy authorizations, on which we do not need to have details. In this chapter, we use authorizations of the `AC4RDF` model defined in Section 3.1.1 (P. 46). On the other hand, other kind of authorizations may be used instead of Auth. Indeed, our model can be used to define upstream policies in order to generate authorizations that are processable by other models such as [Flouris 2010], which makes the solution modular.

Target is the component which reflects the needed conditions under which a GPolicy is applicable. It is defined as a *propositional logic formula* which is evaluated using the user attributes to check whether the targeted policy is applied or not to the user. The formula uses logical connectors $(\vee, \wedge, \neg)$ between atoms (Atom).

OP is a binary operator used in the Atom definition to compare keys and values. Operators may be mapped to arithmetic operators such as $\underset{\cdot}{=}$, $\underset{\cdot}{\leq}$, $\underset{\cdot}{<}$, $\underset{\cdot}{>}$ and , $\underset{\cdot}{\geq}$. Note that one can define other operations such as "SubStringOf, IsPrefix, $\underset{\cdot}{\sqsupseteq}_{RH}$, . . . ".

Atom is the building block of targets. It allows to make operations over keys and values using operators. Let k, k_1, $k_2 \in$ Key, $v \in$ Value, and op a binary predicate in OP. The Atom component is defined as either : (i) $k\,op\,v$ which compares a Key with a Value, or (ii) $k_1\,op\,k_2$ which compares a Key with a Key.

Example 5.2.2 *The following are examples of atoms:*
$age \underset{\cdot}{\geq} 18$: applies to users whose age is greater than or equal to 18.
$role \underset{\cdot}{=} nurse$: applies to users in role nurse.
$role \underset{\cdot}{\sqsupseteq}_{RH}$ doctor: applies to role doctor and all its senior roles, in case of role hierarchy (2.2.1.3 (P. 28)). Here $\underset{\cdot}{\sqsupseteq}_{RH}$ is interpreted by $\sqsupseteq_{RH}$, the hierarchical relation between roles.

Example 5.2.3 *The policy of Example 5.0.1 is interpreted in Table 5.2, built using the formal language shown in Table 5.1. The global policy* P *has two childs : a universal authorization authorization a_u and another policy* P_0*.* P_0*,*

Table 5.2: Example of a global policy P

Policies
$\mathsf{P} \ = [(\top, \mathsf{P}_0), (\top, \mathscr{a}_u)]$
$\mathsf{P}_0 = [(\mathsf{T}_{\mathsf{admin\_staff}}, \mathsf{P}_1), (\mathsf{T}_{\mathsf{doctor} \vee \mathsf{nurse}}, \mathsf{P}_2)]$
$\mathsf{P}_1 = [(\top, \mathscr{a}_4), (\mathsf{T}_{\mathsf{work\_time}}, \mathscr{a}_3), (\mathsf{T}_{\mathsf{work\_time}}, \mathscr{a}_1)]$
$\mathsf{P}_2 = [(\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}}, \mathscr{a}_2), (\top, \mathsf{P}_3)]$
$\mathsf{P}_3 = [(\top, \mathscr{a}_1), (\top, \mathscr{a}_6), (\top, \mathscr{a}_5)]$
Targets
$\mathsf{T}_{\mathsf{admin\_staff}} \ = (role \underset{\cdot}{=} \mathsf{admin\_staff})$
$\mathsf{T}_{\mathsf{doctor} \vee \mathsf{nurse}} = ((role \underset{\cdot}{=} \mathsf{doctor}) \vee (role \underset{\cdot}{=} \mathsf{nurse}))$
$\mathsf{T}_{\mathsf{work\_time}} = ((time \underset{\cdot}{\leq} \text{"}\mathsf{17:00}\text{"}) \wedge (time \underset{\cdot}{\geq} \text{"}\mathsf{08:00}\text{"}))$
$\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}} = ((role \underset{\cdot}{=} \mathsf{doctor}) \wedge (service \underset{\cdot}{=} \mathsf{onc}))$
Authorizations
$\mathscr{a}_1 =$ `GRANT` $($`?r ; ?x ; ?y`$)$ `WHERE` $\{($`?p ; :hasRec ; ?r`$)\}$
$\mathscr{a}_2 =$ `GRANT` $($`?r ; ?x ; ?y`$)$ `WHERE` $\{($`?p ; :hasRec ; ?r`$), ($`?p ; :admitted ; :onc`$)\}$
$\mathscr{a}_3 =$ `DENY` $($`?r ; :disease ; ?d`$)$ `WHERE` $\{($`?p ; :hasRec ; ?r`$)\}$
$\mathscr{a}_4 =$ `GRANT` $($`?p ; ?x ; ?y`$)$ `WHERE` $\{($`?p ; rdf :type ; :Patient`$)\}$
$\mathscr{a}_5 =$ `DENY` $($`?r ; ?x ; ?y`$)$ `WHERE` $\{($`?p ; :hasRec ; ?r`$), ($`?p ; :admitted ; :onc`$)\}$
$\mathscr{a}_6 =$ `GRANT` $($`?r ; ?x ; ?y`$)$ `WHERE` $\{($`?p ; :hasRec ; ?r`$), ($`?p ; :admitted ; :onc`$),$ $($`?p ; :condition ; "CRITICAL"`$)\}$
$\mathscr{a}_u =$ `DENY` $($`?x ; ?y ; ?z`$)$

as well as $\mathscr{a}_u$, are targeted to any user. P_0 *has two child policies namely:* P_1 *and* P_2*.* P_1 *is targeted if the user belongs to the administration staff* ($\mathsf{T}_{\mathsf{admin\_staff}}$) *and* P_2 *if the user is a doctor or a nurse* ($\mathsf{T}_{\mathsf{doc} \vee \mathsf{nurse}}$)*.* P_1 *is defined by three authorizations $\mathscr{a}_1$, $\mathscr{a}_3$ and $\mathscr{a}_4$. $\mathscr{a}_1$ and $\mathscr{a}_4$ are applied at working time only (*$\mathsf{T}_{\mathsf{work\_time}}$*), whereas $\mathscr{a}_3$ is always applied.*

P_2 *is targeted if the user is a doctor or a nurse* ($\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}}$)*.* P_2 *has two child policies namely the authorization $\mathscr{a}_2$ and the policy* P_3*. $\mathscr{a}_2$ is applied if the user is a doctor whose service is* onc *(*$\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}}$*), whereas* P_3 *is applied for everyone,* i.e. *doctors and nurses.* P_3 *targets three authorizations $\mathscr{a}_1$, $\mathscr{a}_6$ and $\mathscr{a}_5$ for any user.*

After defining the syntax of our high-level language for an attribute based access control, we show, in the next section, how are the user requests evaluated over policies. To define the evaluation we use the notation $[\![]\!]^x$, where x represents the name of the evaluation function.

5.2.2 Targets evaluation over user requests

As mentioned above, the user request represents a set of key/value pairs that are used to determine her/his applicable policies. This is done by the evaluation of targets on the request. As a target is made of atoms, and the latter are made of keys and values, we start by showing the evaluation of keys/values on a user request, followed by the evaluation of atoms, and finally we show how targets are evaluated.

Key, Value **Evaluation** :
We first define an evaluation tool function $[\![]\!]^{\mathsf{KV}}$: $(\mathsf{Key} \cup^{+} \mathsf{Value}) \times \mathsf{Request} \rightarrow \mathscr{P}(\mathsf{Value})$. The function takes an element from the disjoint union of Key and Value as parameter, and evaluates it over the user request. If the input parameter is a Key, then $[\![]\!]^{\mathsf{KV}}$ returns the set of values corresponding to the key found in the request. If the input parameter is a value, then it returns a singleton containing this value. Formally:

$$\begin{aligned} &[\![k]\!]^{\mathsf{KV}}(rq) = \{v \mid (k, v) \in rq\} \\ &[\![v]\!]^{\mathsf{KV}}(rq) = \{v\} \end{aligned}$$

Example 5.2.4 *Consider the user requests* rq_0, rq_1, rq_2 *defined in Example 5.2.1. We obtain the values of keys as follows :*

$$\begin{aligned} [\![role]\!]^{\mathsf{KV}}(\mathsf{rq}_0) &= \{\mathsf{doctor}, \mathsf{admin\_staff}\} \\ [\![service]\!]^{\mathsf{KV}}(\mathsf{rq}_0) &= \{\mathsf{onc}\} \\ [\![time]\!]^{\mathsf{KV}}(\mathsf{rq}_1) &= \{\text{"}\mathsf{09:00}\text{"}\} \\ [\![role]\!]^{\mathsf{KV}}(\mathsf{rq}_2) &= \{\mathsf{nurse}\} \end{aligned}$$

OP **Evaluation** :
The function $[\![]\!]^{\mathsf{OP}} : \mathsf{OP} \rightarrow (\mathsf{Value} \times \mathsf{Value} \rightarrow \mathsf{Boolean})$ is an interpretation function that maps an operator in OP to a comparison function used to evaluate the atom. For instance $[\![\dot{\geq}]\!]^{\mathsf{OP}}$ is interpreted as a function that compares two values with the arithmetic operator $\geq$.

Atom **Evaluation** :
An atom is made of a binary operator *op* and a couple of keys or a key and a value. We define a function $[\![]\!]^{\mathsf{Atom}}$ that checks whether an atom matches a request. To evaluate an atom, any key and value in the atom is transformed to two sets of values using $[\![]\!]^{\mathsf{KV}}$ and the user request. The result sets are then compared by checking whether there exists a couple of values, in their Cartesian Product, that are evaluated to true using the atom operator. Let k, k_1, k_2 $\in$ Key, v $\in$ Value, rq $\in$ Request and op be a predicate in OP. $[\![]\!]^{\mathsf{Atom}}$: Atom × Request → Boolean is defined as follows:

$$\begin{aligned}
[\![k\, op\, v]\!]^{\mathsf{Atom}}(rq) &= \exists v_1 \in [\![k]\!]^{\mathsf{KV}}(rq), \exists v_2 \in [\![v]\!]^{\mathsf{KV}}(rq);\ v_1\ [\![op]\!]^{\mathsf{OP}}\ v_2 \\
[\![k_1\, op\, k_2]\!]^{\mathsf{Atom}}(rq) &= \exists v_1 \in [\![k_1]\!]^{\mathsf{KV}}(rq), \exists v_2 \in [\![k_2]\!]^{\mathsf{KV}}(rq);\ v_1\ [\![op]\!]^{\mathsf{OP}}\ v_2
\end{aligned}$$

Example 5.2.5 *Consider the requests* rq_0 *and* rq_2 *defined in Example 5.2.1.*

The atom $(role \doteq \mathsf{doctor})$ *is evaluated on* rq_0 *as follows :*
$[\![(role \doteq \mathsf{doctor})]\!]^{\mathsf{Atom}}(\mathsf{rq}_0) = \big(\exists v_1 \in \{\mathsf{doctor}, \mathsf{admin\_staff}\}, \exists v_2 \in \{\mathsf{doctor}\}; v_1 = v_2\big) = true$

The atom $(role \doteq \mathsf{nurse})$ *is evaluated on* rq_2 *as follows :*
$[\![(role \doteq \mathsf{nurse})]\!]^{\mathsf{Atom}}(\mathsf{rq}_2) = \big(\exists v_1 \in \{\mathsf{nurse}\}, \exists v_2 \in \{\mathsf{nurse}\}; v_1 = v_2\big) = true$

In this case, the interpretation of $\doteq$ *is string equality. The results of* $[\![role]\!]^{\mathsf{KV}}(\mathsf{rq}_2)$ *and* $[\![\mathsf{nurse}]\!]^{\mathsf{KV}}(\mathsf{rq}_2)$ *are obtained by applying (*Key, Value*) evaluation which returns* nurse *for both key and value. Hence the atom is evaluated to true.*

Target **Evaluation** :
The global policy targets are evaluated over user requests to check whether or not the targeted policy is applicable to the user. To evaluate Target, we use the propositional logic based on (Key,Value) structure. We define the function $[\![]\!]^{\mathsf{TR}}$: Target × Request → Boolean which checks whether a target matches a given request.
Let t, t_1, t_2 be a Target elements, rq be a Request and at be an Atom. Then, the evaluation of Target is as follows :

$$\begin{aligned}
[\![t_1 \wedge t_2]\!]^{\mathsf{TR}}(rq) &= [\![t_1]\!]^{\mathsf{TR}}(rq) \wedge [\![t_2]\!]^{\mathsf{TR}}(rq) \\
[\![t_1 \vee t_2]\!]^{\mathsf{TR}}(rq) &= [\![t_1]\!]^{\mathsf{TR}}(rq) \vee [\![t_2]\!]^{\mathsf{TR}}(rq) \\
[\![\neg t]\!]^{\mathsf{TR}}(rq) &= \neg [\![t]\!]^{\mathsf{TR}}(rq) \\
[\![at]\!]^{\mathsf{TR}}(rq) &= [\![at]\!]^{\mathsf{Atom}}(rq) \\
[\![\top]\!]^{\mathsf{TR}}(rq) &= true
\end{aligned}$$

Target component can be expressed in several forms of propositional logic using logical connector $\vee$, $\wedge$, $\neg$ with usual semantics. We evaluate each Atom which is the smallest component of Target component, using $[\![]\!]^{\mathsf{Atom}}$ defined in 5.2.2, then, we combine its results using the classical propositional logic evaluation. $\top$ is always evaluated to *true*.

Example 5.2.6 *Consider the target* $\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}}$ *in Table 5.2, and the request* rq_0 *in Example 5.2.1. The evaluation of* rq_0 *over* $\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}}$ *is performed as follows:*

$$[\![((role \mathrel{\underset{\cdot}{=}} \mathsf{doctor}) \wedge (service \mathrel{\underset{\cdot}{=}} \mathsf{onc}))]\!]^{\mathsf{TR}}(\mathsf{rq}_0) =$$
$$[\![(role \mathrel{\underset{\cdot}{=}} \mathsf{doctor})]\!]^{\mathsf{Atom}}(\mathsf{rq}_0) \wedge [\![(service \mathrel{\underset{\cdot}{=}} \mathsf{onc})]\!]^{\mathsf{Atom}}(\mathsf{rq}_0) = true \wedge true = true.$$

Using the Atom *evaluation defined in Section 5.2.2, and the truth table of logical connector* $\wedge$*, the function will return true. In other words* $\mathsf{T}_{\mathsf{doctor} \wedge \mathsf{onc}}$ *matches* rq_0*.*

5.3 User policy generation

In this section, we show how does the `User Policy Generator` proceed to construct a user policy. Our purpose is to define a global policy which is queried by user requests to finally generate the pair $(\mathscr{A}_s, \mathsf{ch})$ that will be processed by `AC4RDF`. In our illustrative examples, we will suppose that a nurse is requesting access to the triplestore. We will use the request $\mathsf{rq}_2 = \{(role, \mathsf{nurse})\}$ of Example 5.2.1.

The first step of policy evaluation consists in computing the user's applicable authorizations. This is done by computing the user's *subpolicy* according to her/his attributes. The authorizations are computed with respect to the user subpolicy tree by performing a depth traversal of the tree and selecting the leaves *i.e.* authorizations. The next step consists in selecting a ch function that will be used to resolve conflicts. Figure 5.3 illustrates the user policy generation steps.

5.3.1 User SubPolicy

The first step involves the generation of the user subpolicy represented by a subtree of the global policy tree according to the user attributes given in a request. The subtree is generated by traversing the global policy tree in depth,

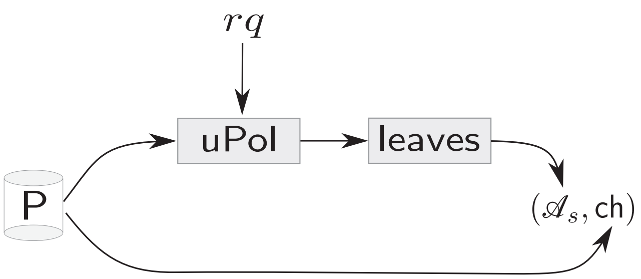


Figure 5.3: User policy generation

and keeping the nodes connected by edges tagged with targets which match the user attributes found in the request. The user policy evaluation is given by the function $\mathsf{uPol} : \mathsf{Request} \times \mathsf{GPolicy} \rightarrow \mathsf{GPolicy}$ which maps a recursive policy to a recursive policy according to a given Request rq. uPol is defined as follows:

$$\mathsf{uPol}(rq)(a) = a$$
$$\mathsf{uPol}\big(rq\big)\big([(t_0, p_0), \ldots, (t_n, p_n)]\big) = \big[\big(t_i, \mathsf{uPol}(rq)(p_i)\big) \mid [\![t_i]\!]^{\mathsf{TR}}(rq)\big]$$

uPol starts from the root policy and performs a recursive traversing of the global policy tree returning a subpolicy tree in which edges are labeled with targets evaluated to true using the user request.

Example 5.3.1 *Suppose a nurse is requesting access over the policy* P *defined in Table 5.2. Her request is translated by* rq_2 *of the Example 5.2.1.* uPol *will give the following subpolicy tree illustrated by Figure 5.4:*
$\mathsf{P} = [(\top, \mathsf{P}_0), (\top, a_u)]$
$\mathsf{P}_0 = [(\mathsf{T}_{\mathsf{doctor} \vee \mathsf{nurse}}, \mathsf{P}_2)]$
$\mathsf{P}_2 = [(\top, \mathsf{P}_3)]$
$\mathsf{P}_3 = [(\top, a_1), (\top, a_6), (\top, a_5)]$

5.3.2 User authorizations selection

Before showing how to select the user subset of authorizations $\mathscr{A}_s$ of the pair $(\mathscr{A}_s, \mathsf{ch})$, we need to define a function that returns the authorizations of a given policy, in other words it returns all the leaves of the policy tree.

$\mathsf{leaves} : \mathsf{GPolicy} \rightarrow \mathscr{P}(\mathsf{Auth})$ is the function which maps a recursive policy to a set of authorizations. The function performs a depth first traversing of the tree and returns the leaves (authorizations). leaves is defined as follows :

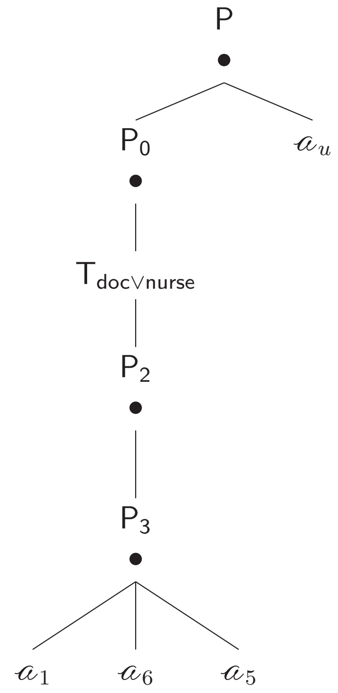


Figure 5.4: Nurse subpolicy tree

$$\begin{aligned}&\mathsf{leaves}(a) = \{a\}\\&\mathsf{leaves}\big((\_, [(\_, \mathsf{p_0}), \ldots, (\_, \mathsf{p_n})])\big) = \textstyle\bigcup_{i\in 0..n}(\mathsf{leaves}(\mathsf{p_i}))\end{aligned}$$

leaves starts from the root policy and makes a recursive call for all child policies. If the policy passed as parameter is not an authorization, then all its child policies are aggregated, whereas if it is an authorization, then it will be returned.

Now we have all the ingredients to define the function $[\![]\!]^{\mathsf{uAuth}}$ which returns the authorizations for a given user request. The function is constructed by the composition of the two functions uPol and leaves. In other words uPol is applied first to get the user policy, then leaves is applied on the resulted policy to get the user authorizations. Formally $[\![]\!]^{\mathsf{uAuth}} : \mathsf{Request} \times \mathsf{GPolicy} \to \mathscr{P}(\mathsf{Auth})$ is defined as follows:

$$[\![rq]\!]^{\mathsf{uAuth}}(p) = \big(\mathsf{leaves} \circ \mathsf{uPol}\big)(rq, p)$$

$[\![]\!]^{\mathsf{uAuth}}$ generates the set of user authorizations $\mathscr{A}_s$ of the pair $(\mathscr{A}_s, \mathsf{ch})$ which will be sent to AC4RDF to be enforced. Remind that the user's policy $(\mathscr{A}_s, \mathsf{ch})$ must respect the *Totality* well-formedness condition by the requirement that $\mathscr{A}_s$ must contain a universal authorization a_u (Lemma 3.2.2 (P. 54)). For simplicity, a_u is assumed to be unique (Assumption 3.2.3 (P. 55)). Hence $[\![]\!]^{\mathsf{uAuth}}$ result must contain a_u for any user's request. To achieve this condition, the root of the global policy should have a child policy a_u with a target $\top$ which is always evaluated to true. That way, a_u will take part of the result of any user's request.

Example 5.3.2 *Consider the last example 5.3.1 where we applied* uPol *to get the user policy. To select the user's authorizations, the function* $[\![]\!]^{\mathsf{uAuth}}$ *applies* leaves *on the result of* uPol *shown in Figure 5.4 as follows:*

$$\begin{aligned}\mathsf{leaves(P)} &= \mathsf{leaves(P_0)} \cup \mathsf{leaves}(\mathpzc{a}_{\mathsf{u}}) \\ &= \mathsf{leaves(P_2)} \cup \{\mathpzc{a}_u\} \\ &= \mathsf{leaves(P_3)} \cup \{\mathpzc{a}_u\} \\ &= \{\mathpzc{a}_1, \mathpzc{a}_6, \mathpzc{a}_5, \mathpzc{a}_u\}\end{aligned}$$

After receiving a user's request rq over the global policy P from the `Policy Enforcer`, the `User Policy Generator` answers with the user's policy $([\![rq]\!]^{\mathsf{uAuth}}(\mathsf{P}), \mathsf{ch})$. ch is parameter that represents the conflict resolution function presented in Section 3.2 (P. 53). In practice, ch parameter can be the name of the Java class that implements the function. The `Policy Enforcer` sends the received couple $([\![rq]\!]^{\mathsf{uAuth}}(\mathsf{P}), \mathsf{ch})$ as well as the user query Q to `AC4RDF` to be enforced. Finally, `AC4RDF` evaluates the query over the positive subgraph, and the result is sent back to the user. The following is an example of how `AC4RDF` generates the nurse positive subgraph.

Example 5.3.3 *Following our examples where a nurse is requesting access, the* `User Policy Generator` *answers to the* `Policy Enforcer` *with the user policy* $\big(\{\mathpzc{a}_1, \mathpzc{a}_6, \mathpzc{a}_5, \mathpzc{a}_u\}, \mathsf{ch}\big)$ *where* ch *applies the* `MSTP` *strategy. Consider the following closed graph G where every triple is assigned the set applicable authorizations belonging to the user nurse:*

	Subject	Predicate	Object	$\mathsf{ar}(\mathsf{Cl}(G), \mathscr{A}_s)(et_i)$	ch
et_1	`:alice`	`:hasRecord`	`:r1`	$\{\mathpzc{a}_u\}$	$\mathpzc{a}_u$
et_2	`:chuck`	`:hasRecord`	`:r2`	$\{\mathpzc{a}_u\}$	$\mathpzc{a}_u$
et_3	`:r1`	`:disease`	`:d1`	$\{\mathpzc{a}_1, \mathpzc{a}_5, \mathpzc{a}_u\}$	$\mathpzc{a}_5$
et_4	`:r2`	`:disease`	`:d2`	$\{\mathpzc{a}_1, \mathpzc{a}_u\}$	$\mathpzc{a}_1$
et_5	`:alice`	`:admitted`	`:onc`	$\{\mathpzc{a}_u\}$	$\mathpzc{a}_u$

Since we want to apply `MSTP` *strategy, a total order will be computed using Algorithm 2 which will give the following result :* $[\mathpzc{a}_3, \mathpzc{a}_4, \mathpzc{a}_6, \mathpzc{a}_2, \mathpzc{a}_5, \mathpzc{a}_1, \mathpzc{a}_u]$.

The triple et_3 is about record `:r1` *which belongs to the patient alice who is admitted to oncology, hence it should not be granted to nurses. For the user nurse, the set applicable authorizations to et_3 is :* $\mathsf{ar}(G, \mathscr{A}_s)(et_3) = \{a_1, a_5, a_u\}$. ch *function will select the smallest authorization a_5. Thus et_3 will be denied as stated by the policy. By applying a similar processing on all triples in G, we obtain $G^+ = \{et_4\}$. The policy states that doctors working in oncology service should be granted access to triple et_3. In case a doctor working in oncology requests access to G, the generated user policy is $\big(\{a_1, a_2, a_6, a_5, a_u\}, \mathsf{ch}\big)$. The set of applicable authorizations to et_3 is* $\mathsf{ar}(G, \mathscr{A})(et_3) = \{a_1, a_2, a_5, a_u\}$. ch *function will select the smallest authorization a_2 and et_3 will be granted.*

5.4 Conclusion

In this chapter we gave the syntax and semantics of an expressive high level language which allows the definition of global policies for multiple users. The global policies are defined by means of trees which intermediate nodes represent other subpolicies and leaves represent authorizations. Subpolicies are accessible through targets which represent conditions over the user's attributes that allow to check whether or not a policy is applicable to the user. The users' requests are evaluated over global policies by the `User Policy Generator` which constructs the user policy. The latter is then enforced by `AC4RDF` which generates a positive subgraph, over which the user's query is evaluated.

Note that we have a global policy from which we computed a part that depends on the user's request ($\mathscr{A}_s$), and a second part that does not depend on the user's request (ch). Hence it is fixed and could be coded in the RDF store once and for all for performance reasons. Then, to link the two parts, it suffices to make an *and* between the user's authorizations and those applicable to the triple. In the next chapter, we propose an enforcement approach for `AC4RDF` model where every triple t in the base graph is annotated with a subset of the global policy authorization set, representing the applicable authorizations $\mathsf{ar}(\mathsf{Cl}(G), \mathscr{A})(t)$. Since the triples are already annotated with their applicable authorizations of the global policy, an upstream operation is needed before applying ch. In fact, given a triple t, ch must be applied on the authorizations applicable to the triple which are also assigned to the user, formally $\mathscr{A}_s \cap \mathsf{ar}(\mathsf{Cl}(G), \mathscr{A})(t)$.

Chapter 6

Policy enforcement and Experiments

Contents

▷ *As mentioned in Chapter 3, **AC4RDF** model semantics are defined with the set of authorizations that belongs to the current subject. In a multi-subject policy context with a set of authorizations $\mathscr{A}$, a subset $\mathscr{A}_s \subseteq \mathscr{A}$ of authorizations is associated to each subject s who executes a (SPARQL) query. In the previous chapter, we showed how this association could be performed using an attribute based approach to generate the subject's policy $(\mathscr{A}_s, \mathsf{ch})$. The latter is then enforced by **AC4RDF** which computes the positive subgraph of the authenticated subject. In this chapter, we show how to enforce multi-subject policies using a data-annotation approach. The idea is to materialize every triple's applicable authorizations of the global policy, into a bitset which is used to annotate the triple. The base graph G is materialized into a graph $G^{\mathscr{A}}$ by annotating every triple $t \in G$ with a bitset representing $\mathsf{ar}(G, \mathscr{A})(t) \subseteq \mathscr{A}$. The subjects are similarly assigned a bitset which represents the set of authorizations assigned to them. When a subject sends a query, the system evaluates it over the her/his positive subgraph computed by **AC4RDF**. We start by giving the semantics of the enforcement approach, then we give the implementation details with experiments.* ◁

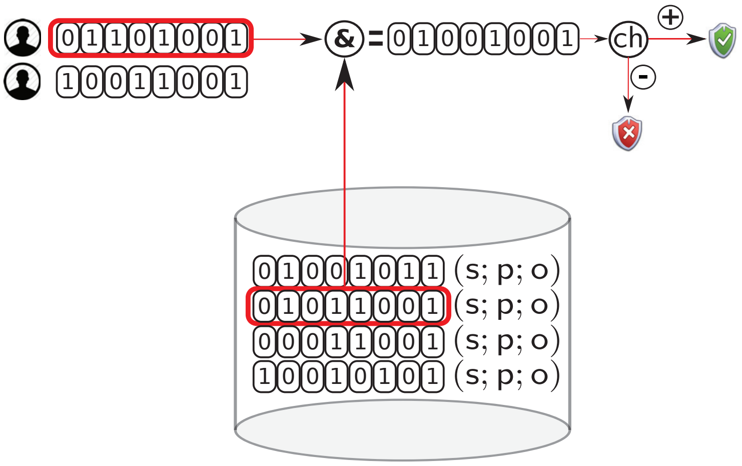


Figure 6.1: Policy Enforcement

6.1 Policy enforcement

As we mentioned in Section 2.3.1 (P. 35), the policy enforcement may be classified into pre-processing, post-processing or annotation-based. The advantage of the pre-processing approaches such as query rewriting techniques, is that the policy enforcer is independent from RDF data. In other words, any updates on data would not affect the policy enforcement. On the other hand, this technique fully depends on the query language. In the post-processing approaches, the query response time may be considerably longer since policies are enforced after all data (allowed and not allowed) have been processed. In the annotation approach, since the triples are already annotated with the access information, only the triples with a grant access can be used to answer the query. On the other hand, any updates in the data would require the re-computation of annotations. Some works [Papakonstantinou 2012] support incremental re-computation of the annotated triples after updates.

To enforce `AC4RDF` model, we use a data-annotation approach which materializes the applicable authorizations in an annotated graph denoted by $G^{\mathscr{A}}$. The latter is computed once and for all subjects at design time. The subjects queries are evaluated over the annotated graph with respect to their assigned authorizations. As illustrated by Figure 6.1, the triple's applicable authorizations as well as the subject's assigned ones are encoded into *bitsets* over which a logical *and* is performed to check whether the triple is authorized or not. In the following, we show how the base graph triples are annotated then we show how the subjects queries are evaluated.

6.1.1 Graph annotation

From a conceptual point of view, an annotated triple can be represented by adding a fourth component hence obtaining a quad. From a physical point of view, the annotation can be stored in the graph name of the SPARQL dataset (Definition 2.1.11 (P. 20)). To annotate the base graph, we use the graph name IRI of the dataset to store bitsets representing the applicable authorizations of each triple. Doing so, we do not need extra mechanisms to enforce our model, in contrast to other approaches which use specific language to interrogate the annotated triples [Lopes 2010]. Given a set of authorizations $\mathscr{A}$, each triple in the base graph is annotated with a bitset of size $|\mathscr{A}|$ in which the bits set to one are those in the position of the syntactical order of the authorizations applicable to the triple. In other words, given an authorization a_i in the input set, the i-th bit is set to 1 in the generated bitset. First we need a bijective function $\mathsf{authToBs}$ which maps a set of authorizations to an IRI representing the bitset. $\mathsf{authToBs}^{\text{-1}}$ is the inverse function of $\mathsf{authToBs}$.

Next we define a function arsg which takes a set of authorizations $\mathscr{A}'$ and a graph G, and returns the subgraph containing triples which the set of applicable authorizations is $\mathscr{A}'$. The function arsg is formally defined as follows:

$$\mathsf{arsg}(\mathscr{A}', G) = \{t \in G \mid \mathsf{ar}(G, \mathscr{A})(t) = \mathscr{A}'\}$$

Example 6.1.1 *Consider the Table 3.2 (P. 51) containing the applicable authorizations w.r.t policy $P = (\mathscr{A}, \mathsf{ch})$ defined in Example 3.1.8 (P. 50).*

$$\mathsf{authToBs}(\{a_1, a_9\}) = 100000001$$
$$\mathsf{arsg}(\{a_9\}, \mathsf{Cl}(G_0)) = \{et_2, et_3, it_3\}$$

Now we are ready to define the annotated graph represented by a dataset (Definition 2.1.11): a set of named graphs in which the name is the binary encoding of $\mathscr{A}'$ and the graph is the nonempty set of triples $\mathsf{arsg}(\mathscr{A}', G)$.

Definition 6.1.2 (Annotated graph) *Given a set of authorizations $\mathscr{A}$ and a graph $G \in \mathsf{BGP}$, the dataset that represents the annotated graph denoted by $G^{\mathscr{A}}$, is defined as the following:*

$$G^{\mathscr{A}} = \big\{\langle u, \mathsf{arsg}(\mathscr{A}', G)\rangle \quad \mid \mathscr{A}' \in \mathscr{P}(\mathscr{A}) \text{ and } \mathsf{arsg}(\mathscr{A}', G) \neq \emptyset \text{ and } u = \mathsf{authToBs}(\mathscr{A}')\big\}$$

Note that having $\mathscr{A}' \in \mathscr{P}(\mathscr{A})$ raises the problem of combinatorial explosion. This issue is discussed in Section 6.3

Example 6.1.3 *Consider the Table 3.2 (P. 51) containing the applicable authorizations w.r.t policy $P = (\mathscr{A}, \mathsf{ch})$ defined in Example 3.1.8 (P. 50). The dataset representing the annotated graph is illustrated by Table 6.1.*

Table 6.1: Example of a dataset representing an annotated graph

u	G
000000111	$\{et_1\}$
000000001	$\{et_2, et_3, it_3\}$
100000001	$\{et_4\}$
001000001	$\{et_5\}$
000100001	$\{et_6\}$
010000011	$\{it_1\}$
000011001	$\{it_2\}$

Definition 6.1.2 defines how to annotate the base graph G given the policy authorizations set. The following lemma ensures that $G^{\mathscr{A}}$ forms a partition of the base graph G.

Lemma 6.1.4 *Given an annotated graph $G^{\mathscr{A}} = \{\langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$, the following properties hold:*

- $\forall i, j \in 1..n : i \neq j \implies G_i \cap G_j = \emptyset$
- $\bigcup_{i \in 1..n} G_i = G$

Proof In the following, we prove that $\forall i, j \in 1..n : i \neq j \implies G_i \cap G_j = \emptyset$.

We prove that $\forall i, j \in 1..n : G_i \cap G_j \neq \emptyset \implies i = j$. $G_i \cap G_j \neq \emptyset$ means that there exists t such that $t \in G_i$ and $t \in G_j$. Since $t \in G_i$ then $u_i = \mathsf{authToBs}(\mathsf{ar}(G, \mathscr{A})(t))$. Similarly, since $t \in G_j$ then $u_j = \mathsf{authToBs}(\mathsf{ar}(G, \mathscr{A})(t))$. Which means that $u_i = u_j$, hence $i = j$.

In the following we prove that $\bigcup_{i \in 1..n} G_i = G$

First we prove that $\forall t \in \bigcup_{i \in 1..n} G_i \implies t \in G$.

$\forall t \in \bigcup_{i \in 1..n} G_i$ means that $\exists \langle u, G' \rangle \in G^{\mathscr{A}} \mid t \in G'$. Since by definition, $G' \subseteq G$ then $t \in G$.

We prove that $\forall t \in G \implies t \in \bigcup_{i \in 1..n} G_i$.

Since $\mathscr{a}_u \in \mathscr{A}$ then $\forall t \in G$, $\mathsf{ar}(G, \mathscr{A})(t) \neq \emptyset$, hence $\exists \langle u', G' \rangle \in G^{\mathscr{A}}$ s.t. $t \in G'$. Which means that $t \in \bigcup_{i \in 1..n} G_i$. ∎

6.1.2 User's query evaluation

Following Definition 3.1.9, given a global policy authorization set $\mathscr{A}$, the positive subgraph of a subject having $\mathscr{A}_s \subseteq \mathscr{A}$ as applicable authorizations, is given by the following : $G_s^+ = \{t \in G \mid (\mathsf{effect} \circ \mathsf{ch} \circ \mathsf{ar}(G, \mathscr{A}_s)(t) = +\}$. Since we materialized the set of applicable authorizations in $G^{\mathscr{A}}$, we need to define the subject's positive subgraph from the graph annotation, more precisely from $\mathsf{ar}(G, \mathscr{A})$. The following lemma shows that $\mathsf{ar}(G, \mathscr{A}_s)$ can be computed from $\mathscr{A}_s$ and $\mathsf{ar}(G, \mathscr{A})$.

Lemma 6.1.5 *Given a graph G, a set of policy authorizations $\mathscr{A}$ and a subset of subject authorizations $\mathscr{A}_s$, the following holds for any $t \in G$:*

$$\mathscr{A}_s \cap \mathsf{ar}(G, \mathscr{A})(t) = \mathsf{ar}(G, \mathscr{A}_s)(t)$$

Proof By Definition 3.1.3 (P. 47), $\mathsf{ar}(G, \mathscr{A})(t) = \{\mathscr{a} \in \mathscr{A} \mid \exists \theta \in [\![\mathsf{hb}(\mathscr{a})]\!]_G . t = (\mathsf{head}(\mathscr{a}))\theta\}$, hence $\mathscr{A}_s \cap \mathsf{ar}(G, \mathscr{A})(t) = \{\mathscr{a} \in \mathscr{A}_s \cap \mathscr{A} \mid \exists \theta \in [\![\mathsf{hb}(\mathscr{a})]\!]_G . t = (\mathsf{head}(\mathscr{a}))\theta\}$. Since $\mathscr{A}_s \subseteq \mathscr{A}$ then $\mathscr{A}_s \cap \mathscr{A} = \mathscr{A}_s$. Which means that $\mathscr{A}_s \cap \mathsf{ar}(G, \mathscr{A})(t) = \{\mathscr{a} \in \mathscr{A}_s \mid \exists \theta \in [\![\mathsf{hb}(\mathscr{a})]\!]_G . t = (\mathsf{head}(\mathscr{a}))\theta\} = \mathsf{ar}(G, \mathscr{A}_s)(t)$.

Similarly to the triples, subjects are assigned bitsets representing authorizations applicable to them. If a subject authorization set is $\mathscr{A}_s$, then he is assigned the bitset *ubs* where the i-th bit is set to one if $\mathscr{a}_i \in \mathscr{A}_s$.

Example 6.1.6 *Given the set of authorizations $\mathscr{A}$ in Table 3.1 (P. 47) Eve is a nurse who can see the patients having tumors ($\mathscr{a}_1$) and which service the patients are admitted to ($\mathscr{a}_6$). She is denied anything else ($\mathscr{a}_9$). Her assigned bitset is the following:* 100001001. *Dave belongs to the administrative staff, he can access doctors services assignment ($\mathscr{a}_3$) and the patients they treat ($\mathscr{a}_4$). He is denied anything else ($\mathscr{a}_9$). His assigned bitset is the following:* 001100001.

Once the graph is annotated, it is made available to the subjects with a filter function which prunes out the inaccessible triples given the subject's authorization set. In other words, the filter function returns the subjects's positive subgraph by applying the ch function on the subject's assigned authorizations $\mathsf{ar}(G, \mathscr{A}_s)(t)$. We showed in Lemma 6.1.5 that this subset can be obtained from the applicable authorizations in $G^{\mathscr{A}}$ by computing a bitwise logical *and* (denoted by &) between the subjects' bitsets and triples' bitsets, as illustrated in Figure 6.1.

Table 6.2: Example of annotated graph and users bitsets

user	*ubs*
Eve	100001001
Dave	001100001

$\mathsf{Cl}(G_0)^{\mathscr{A}}$		$ubs\ \ \&\ \ u$	
u	G	Eve	Dave
000000111	$\{et_1\}$	000000001	000000001
000000001	$\{et_2, et_3, it_3\}$	000000001	000000001
100000001	$\{et_4\}$	100000001	000000001
001000001	$\{et_5\}$	000000001	001000001
000100001	$\{et_6\}$	000000001	000100001
010000011	$\{it_1\}$	000000001	000000001
000011001	$\{it_2\}$	000001001	000000001

Definition 6.1.7 *Given a subject's bitset ubs and an annotated graph $G^{\mathscr{A}}$,* filter *is defined as follows:*

$$\mathsf{filter}(G^{\mathscr{A}})(ubs) = \bigcup\{G_i \mid \langle u_i, G_i\rangle \in G^{\mathscr{A}} \wedge \\ (\mathsf{effect} \circ \mathsf{ch} \circ \mathsf{authToBs}^{-1}(u_i\ \ \&\ \ ubs) = +\}$$

Once the subject's positive subgraph is computed with filter function, the subject's query Q is then evaluated over it returning $[\![Q]\!]_{\mathsf{filter}(G^{\mathscr{A}})(ubs)}$ to the subject.

Example 6.1.8 *Let us consider the policy $P = (\mathscr{A}, \mathsf{ch})$ of Example 3.1.8 (P. 50). Table 6.2 illustrates the annotated graph obtained from $\mathsf{Cl}(G_0)$ shown in Figure 2.2 (P. 16), as well as the two users of Example 6.1.6 with their assigned authorizations. The* filter *function computes the positive subgraph of Eve as follows: $\mathsf{filter}(G_0^{\mathscr{A}})(100001001) = \{et_4, et_8\}$. Similarly, Dave's positive subgraph equals $\{et_5, et_6\}$.*

6.2 Implementation

Our system is implemented using the Jena Java API on top of the Jena TDB[1] (quad) store. Apache Jena is an open source Java framework which provides an API to manage RDF data. ARQ[2] is a SPARQL query engine for Jena which allows querying and updating RDF models through the SPARQL standards. ARQ supports custom aggregation and `GROUP BY`, `FILTER` functions and path queries.

[1] https://jena.apache.org/documentation/tdb/

[2] https://jena.apache.org/documentation/query/

Algorithm 4 Quad filter

```
Input: processed quad $(s, p, o, u)$ and subject's bitset $ubs$
Output: $Boolean$
  function QuadFilter($(s, p, o, u), ubs$)
      Let $bs \leftarrow u \;\&\; ubs$
      Let $decision \leftarrow bsDecisionCache.get(bs)$
      if $decision = null$ then
          $decision \leftarrow (\mathsf{effect} \circ \mathsf{ch} \circ \mathsf{authToBs}^{-1}(bs) = +)$
          $bsDecisionCache.set(bs, decision)$
      end if
      return $decision$
  end function
```

Jena TDB is a native RDF store which allows to store and query RDF triples. Jena TDB follows the SPARQL standard by supporting quads that are stored in datasets. Jena provides rules engines to derive additional RDF triples from those that are explicitly defined. To be able to support inference, our access control model needs a forward chaining reasoner. In the presence of inference, both explicit and implicit triples will be annotated which means that the closure of the base graph is computed before the annotation operation. To generate $G^{\mathscr{A}}$, the dataset of annotated triples, we use SPARQL `CONSTRUCT` queries to obtain authorizations scopes (see Definition 3.1.5 (P. 48)). An authorization $\mathscr{a}$ is transformed into $Q_{\mathscr{a}}$ = `CONSTRUCT` $\mathsf{head}(\mathscr{a})$ WHERE $\mathsf{hb}(\mathscr{a})$. We use an in-memory hash map in which we store the *ids* of the triples and the correspondent bitset. For every authorization $\mathscr{a}_i$, a `CONSTRUCT` query $Q_{\mathscr{a}_i}$ is run over the raw dataset, and the result triples are added/updated to the hash map with the bit i set to 1. Once the hash map computed, it is written into a dataset which represents $G^{\mathscr{A}}$. Note that we could have used the dataset directly instead of a hash map, but this would be time consuming due to the high number of disk accesses. During query evaluation, on the fly filtering is applied to the accessed triples. Jena TDB provides a low level quad filter hook[3] that we use for implementation. For each accessed quad, let u be the quad's graph IRI, t its triple and ubs be the subject's bitset. A bitwise logical *and* is performed between (the bitset represented by) u and ubs. The ch function on the authorizations obtained by $\mathsf{authToBs}^{-1}$ is then applied in order to allow or deny access to t. If t is allowed, then it is transmitted to the ARQ engine to be used by query Q. Otherwise, it will be hidden to the ARQ engine. An in-memory cache is used to map quad graph IRIs to grant/deny decisions in order to speedup the filtering process. Algorithm 4 illustrates the quad filter function integrated to Jena TDB to filter the triples.

[3] http://jena.apache.org/documentation/tdb/quadfilter.html

Table 6.3: Summary of notations

in	$\lvert G \rvert$	Size of the LUBM dataset
in	$\lvert \mathscr{A} \rvert$	Number of authorizations
in	$\lvert G^{+} \rvert_{\lvert G \rvert}$	Positive subgraph size w.r.t raw dataset size
in	$\lvert \mathscr{A}_s \rvert$	Number of authorizations assigned to the user
in	Q_s	LUBM test Query
out	t_A	Time to build $G^{\mathscr{A}}$ in memory
out	t_W	Time to write $G^{\mathscr{A}}$ to disk
out	$t_{G^{+}}$	Time to evaluate Q on materialized G^{+}
out	$t_{G^{\mathscr{A}}}$	Time to evaluate Q on $G^{\mathscr{A}}$
out	t_G	Time to evaluate Q on (raw) G

6.2.1 Experiments

The key input factors for the benchmarking of our solution are the sizes of the *base graphs*, the sizes of the *access control policies*, the sizes of positive subgraphs, the sizes of users policies and the user queries. The factors are reported in Table 6.3. The *base graphs* are synthetic graphs generated by the Lehigh University Benchmark (LUBM)[4]. Their sizes ($\lvert G \rvert$) vary from 126k to 1,591k triples. The *access control policies* are randomly generated using the LUBM vocabulary (about universities and people therein), with three control parameters. The first control parameter is the number of authorizations ($\lvert \mathscr{A} \rvert$) and varies from 50 to 200 authorizations. The second control parameter is the *scope average* of the policy with respect to the G. In other words, the percentage of triples in G which are under the influence of the policy authorizations. The last control parameter is the size of the body of each (atomic) authorization $\mathscr{a} \in \mathscr{A}$. The results we report here are for fixed scope (about 4% by authorization) and fixed sizes of bodies (set to 2 for each authorization). The size of positive subgraph parameter $\lvert G^{+} \rvert_{\lvert G \rvert}$ varies from 10 to 100% of $\lvert G \rvert$ and the number of user authorizations $\lvert \mathscr{A}_s \rvert$ from 50 to 200. Regarding the user's query parameter Q_s, we used a subset of LUBM test queries. We analyze both the static (creation time) and the dynamic (evaluation time) performance of our solution. Each experiment is run 6 times on 2 cores and 4 GB virtual machines running on OpenStack.

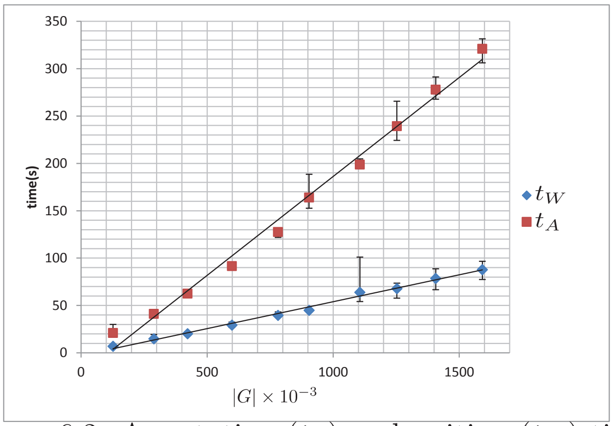


Figure 6.2: Annotation (t_A) and writing (t_W) times

6.2.1.1 Static performance

We distinguish the time needed to compute $G^{\mathscr{A}}$ between the time required for its *building* and the time required for its *writing*. The time to *build* the authorization bitset $\mathsf{ar}(G, t)$ associated with each triple $t \in G$ in memory is referred to as t_A in Table 6.3. The time to *write* the annotated graph $G^{\mathscr{A}}$ from the memory to the quad store is referred to as t_W in Table 6.3. Figure 6.2 shows t_A and t_W with $|\mathscr{A}|$ being set to 100 authorizations. Figure 6.3 shows t_A and t_W with $|G|$ being set to 1,591k triples. As each $\mathscr{a} \in \mathscr{A}$ is mapped to a SPARQL `CONSTRUCT` query, the results show that t_A grows linearly when $|G|$ or $|\mathscr{A}|$ gets bigger. The annotation time is not negligible but we argue that it is not an issue: $G^{\mathscr{A}}$ is computed once, as long as $\mathscr{A}$ is not modified. Figure 6.3 shows that t_A grows linearly when $|\mathscr{A}|$ grows. However, as expected, the results show that t_W is independent of $|\mathscr{A}|$: the overhead incurred by the

[4] http://swat.cse.lehigh.edu/projects/lubm/

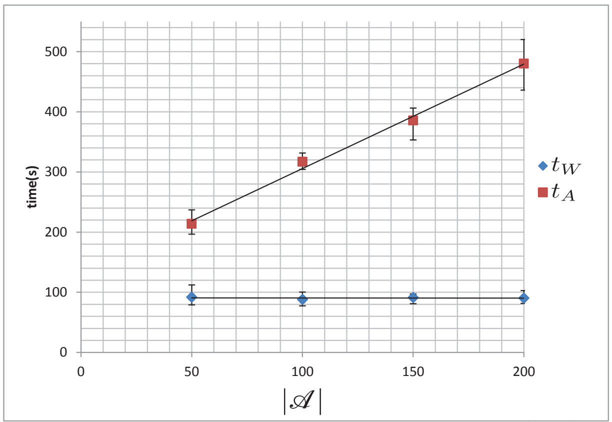


Figure 6.3: Annotation (t_A) and writing (t_W) times

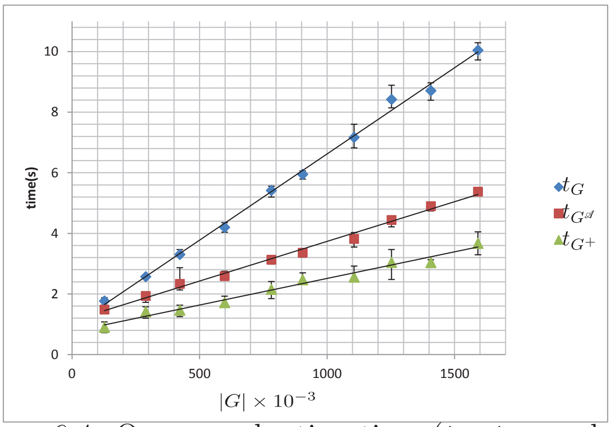


Figure 6.4: Query evaluation time (t_G, $t_{G^{\mathscr{A}}}$ and t_{G^+})

growing size of the bitsets is negligible for $|\mathscr{A}| \in \{50, 100, 150, 200\}$. On average, the annotated graph $G^{\mathscr{A}}$ requires 50% more space than G.

6.2.1.2 Dynamic performance

To evaluate the performance of our solution at runtime, we compare our approach to two extreme methods. Each method computes the positive subgraph G^+ obtained from the base graph G according to a set $\mathscr{A}$ of authorizations.

The first extreme (naive) method gives an upper bound on the overhead incurred by the filtering process. Indeed, in the post-processing approaches, the access control consists in two steps : (1) compute the full answer $Q(G)$ and (2) filter out denied triples from $Q(G)$ as a post-processing step. This method avoids duplication of the base graph G at the price of high overhead at runtime. In our experiments, we considered the step (1) only, by computing

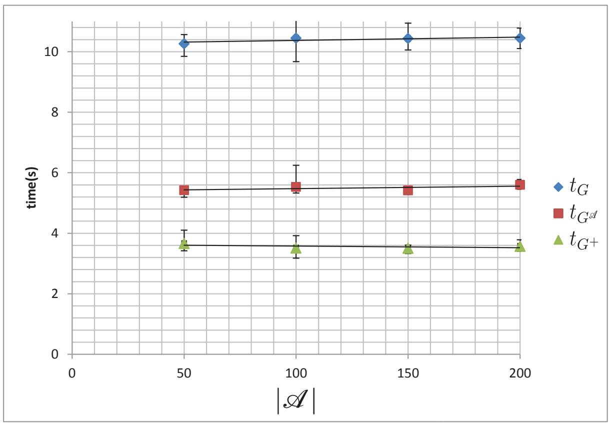


Figure 6.5: Query evaluation time (t_G, $t_{G^{\mathscr{A}}}$ and t_{G^+})

the full answer $Q(G)$. We refer to this method as t_G in Table 6.3.

The second extreme method gives a lower bound on the overhead incurred by the filtering process. The idea is to materialize G^+ for each user profile and then compute $Q(G^+)$. We refer to this method as t_{G^+} in Table 6.3. This method avoids the filtering post-process at the price of massive duplication and storage overhead, as a graph is materialized for each user profile. In contrast, our approach, namely $t_{G^{\mathscr{A}}}$ in Table 6.3, is a trade-off between the extreme ones: it needs some static computation while offering competitive runtime performance. Our results are shown in Figure 6.4 for varying sizes of $|G|$ with $|\mathscr{A}|$ and $|\mathscr{A}_s|$ set to 100, and $|G^+|_{|G|}$ set to 40%. The user's query Q_s is set to the worst case which is the select all query.

The key insight from these experiments is that the overhead *is independent from* $|G|$ and is about 50%.

Another advantage of our approach is its independence from the number of authorizations of both the policy and those assigned to the user. In Figures 6.5 and 6.6 we vary the number of policy authorizations ($|\mathscr{A}|$) and user authorizations $|\mathscr{A}_s|$ respectively, with $|G|$ set to 1,591k triples and Q_s to the select all query. The experiments show a constant overhead while changing $|\mathscr{A}|$ or $|\mathscr{A}_s|$.

Regarding $|G^+|_{|G|}$, the size of the positive subgraph with respect to the size of the annotated graph, the experiments in Figure 6.7 show that the query answer time $t_{G^{\mathscr{A}}}$ grows linearly when $|G^+|_{|G|}$ grows, with $|G|$ fixed to 1,591k and $|\mathscr{A}|$ and $|\mathscr{A}_s|$ fixed to 100. Q_s being the select all query. This shows that the overhead w.r.t. $Q(G^+)$ does not depend on the size of the positive subgraph. Note that t_G does not vary since we did not consider the filtering step of post-processing approaches, otherwise it would grow when $|G^+|_{|G|}$ grows.

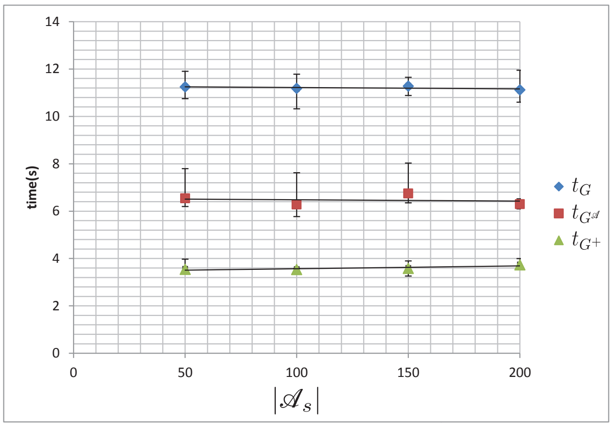


Figure 6.6: Query evaluation time (t_G, $t_{G^{\mathscr{A}}}$ and t_{G^+})

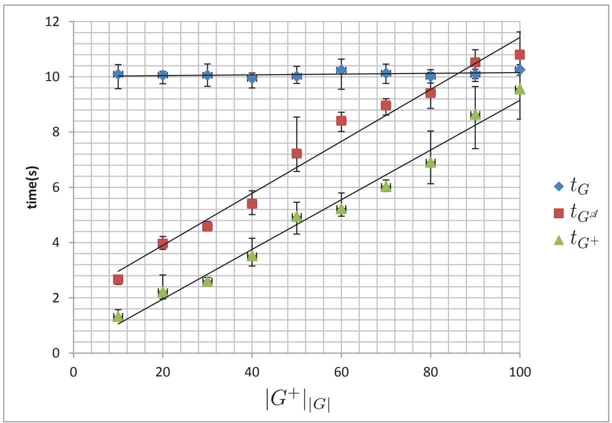


Figure 6.7: Query evaluation time (t_G, $t_{G^{\mathscr{A}}}$ and t_{G^+})

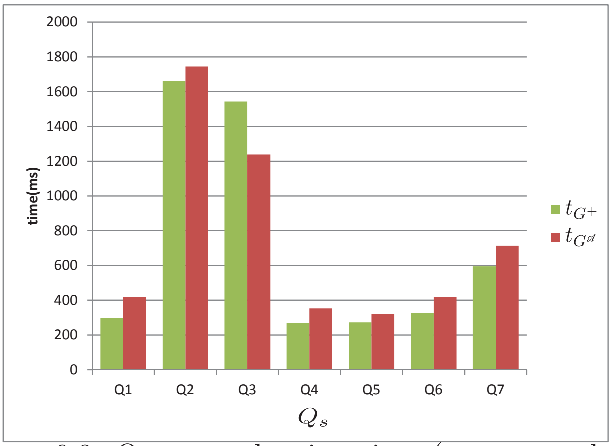


Figure 6.8: Query evaluation time (t_G, $t_{G^{\mathscr{A}}}$ and t_{G^+})

In Figure 6.8 we run experiments on our system with a subset of LUBM test queries used by [Atre 2010], with $|\mathscr{A}|$ and $|\mathscr{A}_s|$ set to 100, and $|G^+|_{|G|}$ set to 40%. We computed the LUBM queries evaluation times and repeated the experiments 100 times. Q1 and Q3 are more complex queries having a high number of initial triples associated with the triple patterns, but the final number of results is quite small (28 and 0 respectively). For these queries, the initial selectivity of the triple patterns and selectivity of the intermediate join results were quite low, but together they gave highly selective results (these queries have cyclic dependency among join variables). Figure 6.8 shows that the time to evaluate query Q3 in presence of the filter $t_{G^{\mathscr{A}}}$ is smaller than the evaluation time over materialized positive subgraph t_{G^+}. The reasons could be the empty result of Q3 or different execution plans.

6.3 Conclusion

In this chapter, we proposed an enforcement framework to the AC4RDF access control model. We used a data annotation approach where the base graph is annotated at the policy design time. Each triple is annotated with a bitset representing its applicable authorizations. The subjects' queries are evaluated over their positive subgraph constructed using her/his bitset and the triples' bitset. The experiments showed that the annotation time is not negligible, but we argue that it is not an issue since this operation is done once and for all during policy design time. Moreover, in Figure 6.2 the ratio t_A/t_W is about 3.4 on average for fixed value of G. Which means that we need 3.4 times of the time to write the base graph to compute in-memory annotation plus the time to write the result to the disk, *i.e.* a total time of 4.4 times of the time to write the base graph. The materialization approach gives fast query answering, however it could lead to a massive duplication and storage overhead, as a graph is materialized for each user profile. Our method is amortized if the sum of triples in the materialized named graphs is approximatively 5 times greater then the number of triples in the base graph. We showed that the overhead of the subject query evaluation is independent from the size of the base graph, and it is about 50%. Moreover, we showed that the query evaluation is independent from the number of policy authorizations as well as the used query language since the way we implemented our access control is independent from the query language as we did not change anything on the ARQ query engine. The annotation bitset represents a combination of authorizations (Definition 6.1.2), may lead to the problem of combinatorial explosion. Theoretically, there is at most $2^{|\mathscr{A}|-1}$ bitsets, since the universal authorization is always applicable. However, some combinations may never happen. For instance, if $\mathscr{a}$ is more specific then $\mathscr{b}$, then for any triple t, if $\mathscr{b}$ is not applicable to t then $\mathscr{a}$ is not applicable to $t \in G$. Hence we will never have a bitset where the position of $\mathscr{b}$ is set to 0 and that of $\mathscr{a}$ set to 1. Furthermore, some authorizations applicability may never apply at the same time. For instance, given two authorizations $\mathscr{a}$ and $\mathscr{b}$ with non-unifiable heads, then they will never apply to the same triple. Hence we will never have a bitset where the positions of $\mathscr{a}$ and $\mathscr{b}$ are set to 1. In addition, in real-life scenarios, the user is assigned a subset of authorizations, which reduces the number of combinations.

Chapter 7

Conclusion

Contents

▷ *In this chapter, we start by giving a summary of the contributions proposed in this thesis, namely :* `AC4RDF` *a fine-grained access control model for RDF, the static verification algorithm, the high-level attribute based language and the enforcement framework. In the next section, we discuss some limitations of the proposed solutions and we give directions for future improvements and future work.* ◁

7.1 Summary

In this thesis, we considered the problem of controlling the access to RDF data in the context of Linked Data. We started by considering the works that have been done in this research field, by defining the comparison criteria, including the expressiveness of policy specification languages, conflict resolution generated by conflicting decisions and the verification of unauthorized inferences. We found out that several criteria have not been well considered. Starting from this fact, we defined the syntax and the semantics of an access control model named AC4RDF allowing a fine grained selective disclosure of RDF data. The semantics of our model are defined by means of the positive subgraph from the base graph, on which the user query is evaluated. This makes our model independent from the query language in contrast to query rewriting techniques. As the real semantics of a graph are represented by its closure, we consider the inferred triples like the explicit ones with respect to authorization policy. In fact, our model grants/denies access to implicit triples if the policy states that it must be granted/denied to the user, in contrast to propagation approaches where some implicit triples could not be returned to the user even though the policy states that she/he is granted access to them.

We showed that inference could be used by a potentially malicious user to access forbidden information, this problem is named the *inference leakage problem*. A user who knows the inference rules, could use a local reasoner over her/his accessible triples, to infer triples she/he is not supposed to access. We formalized this problem into the consistency property which, if respected by the policy, ensures that denied triples can not be inferred once the data have been disclosed to the 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. We proposed a correct an complete algorithm, which, without any knowledge of the base graph, checks if a policy presents an inference leakage problem, in contrast to the works that considered such problem, on the base graph. In case of a policy inconsistency, the algorithm generates counterexample graph patterns that can be used by the administrator to fix the policy or as integrity constraints that do not allow updates leading to inference leakage.

The Attribute Based Access Control Model has shown its flexibility by allowing expressive policies based on the subject's attributes. Given the open nature of the web, we proposed the syntax and semantics of an expressive high level language which allows the definition of global policies for multiple users. The global policies are defined by means of trees which intermediate nodes represent other subpolicies and leaves represent authorizations. Subpolicies are accessible through targets which represent conditions over subject's attributes that define whether or not a policy is applicable to the subject. Subject's requests are evaluated over global policies to construct a subject's policy which is then enforced by `AC4RDF` to generate a positive subgraph, over which the user's query is evaluated.

To enforce our model, we proposed a data-annotation approach where we annotate every triple of the base graph with a bitset representing its applicable authorizations. Similarly, users are assigned bitsets representing the authorizations applicable to them. We proved that our encoding is correct and we gave details about the annotation process as well as the evaluation of the subject's queries. We used the graph name position to store the bitset of applicable authorization of the triple which means that no additional mechanisms are needed to enforce our model, in contrast to approaches that use specific extensions of RDF data model to store annotations. We run experiments of our solution implemented on a concrete RDF store and showed that our implementation incurs reasonable overhead at runtime (about +50%) with respect to the optimal solution which consists in materializing the user's accessible subgraph.

7.2 Discussion and future work

The solutions proposed in this thesis allow a fine-grained selective disclosure of RDF data in the context of Liked Data. Following our requirements on which our proposals were driven (Section 6.2), our future work regards the following directions :

7.2.1 *Expressiveness*

One limitation of our approach is that some real-life policies that depend on the user's profile may require the creation of several authorizations. For instance a policy may state that a doctor can access only the disease of the patients he treats. To apply this policy in the present framework, we should create an authorization for each user's profile and assign it to the user in question. We would like to extend our model with *parameterized authorizations* where the patterns may contain parameters in the subject or object position. To enforce the policies, the parameters are instantiated with terms which represent values of the user's attributes. For instance, the previous policy example would be translated into the authorization : $\mathscr{a}_{id} = \mathtt{GRANT}(?\mathtt{p}\,;\mathtt{:disease}\,;?\mathtt{ds})\,\mathtt{WHERE}\,\{(\mathtt{\$id}\,;\mathtt{:treats}\,;?\mathtt{p})\}$. This authorization is instantiated by replacing `$id` with the IRI representing the doctor's identifier. Hence, adding parameters, leads to changing the semantics of authorizations by introducing a new substitution which replaces parameters with user's attributes. This extension raises new issues related to our enforcement approach. Indeed, our enforcement solution is static, and the problem is that these parametrized authorizations are hard to deal with statically. One solution could be to consider parameters as variables in the annotation operation. However, the user query evaluation process must be extended. Indeed, the ch function may chose a parametrized authorization which once instantiated, may not be applicable to the triple. Hence all the parameterized authorizations in the subset of the triple's applicable authorizations must be dynamically instantiated with the user's attributes to check whether they are applicable or not.

We showed in Chapter 3 how to apply the *Most Specific Takes Precedence* (MSTP) strategy by defining the specificity relation between authorizations. We showed that this strategy has a semantic and syntactic definition. The issue is that they may no longer match when considering the inference rules. Hence, the MSTP strategy would not be fully applied, which decreases the effectiveness of the policy application. To illustrate this, consider the following two authorizations: $\mathscr{a} = \mathtt{GRANT}\,(?\mathtt{d}; :\mathtt{treats}; ?\mathtt{p})\,\mathtt{WHERE}\,\{(?\mathtt{d}; \mathtt{rdf}\,:\mathtt{type}; :\mathtt{Doctor})\}$, and $\mathscr{b} = \mathtt{GRANT}\,(?\mathtt{d}; :\mathtt{treats}; ?\mathtt{p})\,\mathtt{WHERE}\,\{(?\mathtt{p}; :\mathtt{condition}; \mathtt{CRITICAL})\}$. By the definition of the partial order between authorizations (Definition 3.1.14), $\mathscr{a}$ and $\mathscr{b}$ are not comparable. However, if we have a vocabulary which states that :treats domain is :Doctor and its range is :Patient, then the two authorizations become semantically comparable on the closure of any graph G computed using RDFS rules (in particular rules **RDom** and **RRan**), *i.e.* $\mathscr{b}$ is more specific then $\mathscr{a}$. One idea could be to translate schema triples into axiomatic inference rules R' and to add them to set of inference rules R, *i.e.* $\mathsf{R}'' = \mathsf{R} \cup \mathsf{R}'$. Definition 3.1.14 could be extended with R'' where an authorization $\mathscr{a}_1$ is more specific than $\mathscr{a}_2$ if $\exists\theta.\, \mathsf{Cl}_{\mathsf{R}''}(\mathsf{hb}(\mathscr{a}_2))\theta \subseteq \mathsf{Cl}_{\mathsf{R}''}(\mathsf{hb}(\mathscr{a}_1)) \wedge \mathsf{head}(\mathscr{a}_2)\theta = \mathsf{head}(\mathscr{a}_1)$. By this new definition, $\mathscr{a}$ and $\mathscr{b}$ of the above example are comparable, *i.e.* $\mathscr{b}$ is more specific then $\mathscr{a}$.

We plan to extend AC4RDF with update operations. Syntactically, extending our model with updates amounts to add the operation name to the authorizations. For instance, to allow a user to insert patients' admissions, the authorization is as follows : $\mathtt{GRANT\ INSERT}\ (?\mathtt{p}\ ; :\mathtt{admitted}\ ; ?\mathtt{s})$. Semantically, new issues raise from updates, such as policy inconsistencies that stem from implicit insertion/deletion. Indeed, inserting/deleting a triple may lead to the insertion/deletion of other triples. Hence, an inconsistent policy may state that a user is allowed to insert a triple, but she/he she is not allowed to insert some of its consequences that can be inferred. Our static verification algorithm could be adapted to detect such policy inconsistencies by checking if the premises of an inference rule match `GRANT INSERT` authorizations and if its conclusion matches a `DENY INSERT` authorization. Triples' deletion could be treated similarly. Updates also raise a new kind of inference leakage problem linked to the so-called *interference property* [McLean 1990]. A user could send a write query to the system, and if the answer is negative then she/he can infer the existence of triples she/he is not supposed to read.

7.2.2 *Verifiability*

In Chapter 4 (Inference leakage problem and solution), the counterexamples are used to fix the policy manually by the administrator. Updating the policy may lead to the elimination of some counterexamples as well as the generation of new ones. The problem is that when the administrator updates the policy, she/he does not know what are the counterexamples eliminated/-generated. The latter could help the administrator understand the inference leakage problem in her/his policy. We would like to improve the user interaction of the algorithm. When the administrator adds a new authorization, the next run of the algorithm finds the counterexamples affected by this new authorization. If the new added authorization effect is deny, the algorithm checks the inference rules which conclusion unifies with the head of the new authorization. The generated counterexamples from these rules are then displayed to the administrator as the impact of her/his policy modification.

7.2.3 *Performance*

Regarding the static verification Algorithm, we mentioned that the counterexamples can be used as integrity constraints. A high number of counterexamples may burden the query evaluation. Some of the counterexamples may have no sense w.r.t the application domain or w.r.t a given vocabulary. Languages such as ShEx [Prud'hommeaux 2014] enable RDF validation through the declaration of constraints on the RDF model, based on regular expressions. We plan to extend the algorithm to filter out the inconsistent graphs that do not satisfy such constraints. This would reduce considerably, the number of the generated counterexamples.

One future work direction is to optimize the annotation process using the specificity relation between authorizations. If an authorization $\mathcal{a}_1$ is more specific than $\mathscr{a}_2$, then for all t in the base graph, if $\mathscr{a}_1$ is applicable to t then $\mathscr{a}_2$ is applicable as well. We could use this property to optimize the annotation operation since all the triples in the scope of $\mathscr{a}_1$ do not need to be checked for $\mathscr{a}_2$. The annotation algorithm would be as follows : (1) compute the set $\mathsf{mins}_{\sqsubseteq_{\mathtt{MSTP}}}(\mathscr{A})$ containing the most specific authorizations in $\mathscr{A}$ (2) for each bitset of the triples in the scope of $\mathscr{a} \in \mathsf{mins}_{\sqsubseteq_{\mathtt{MSTP}}}(\mathscr{A})$, set the bit of all authorizations $\{\mathscr{b} \mid \mathscr{b} \sqsubseteq_{\mathtt{MSTP}} \mathscr{a}\}$ to 1 (3) remove the subset $\mathsf{mins}_{\sqsubseteq_{\mathtt{MSTP}}}(\mathscr{A})$ from $\mathscr{A}$ (4) repeat (1) to (3) until $\mathscr{A}$ is empty.

The issue of annotations re-computation arises when the annotated graph is updated. Instead of recomputing all the annotations after updates, we could use an incremental annotation, such that, if a triple is inserted/updated/deleted in the annotated graph, only the triples' bitsets affected by this update will be recomputed. Once a triple is inserted/updated/deleted in the annotated graph, the triples' bitsets that might be affected are those which match a pattern in the body of some authorizations. Hence, only some bits should be updated. Such an optimization would reduce the number of updates on the annotated graph.

Another issue concerns the implementation of the annotation algorithm. We used an in-memory cache to store the triples Ids and bitsets temporarily before being stored in the dataset representing the annotated graph. In case of a high number of triples that can't hold in memory, we could use a hybrid approach by loading the triples partially to reduce the disk I/O. One solution could be to select subsets from the set of authorizations $\mathscr{A}$, and computing an in-memory hashmap containing annotated triples with respect to these subsets (see section 6.2 (P. 98)). Once the hashmap is computed, it is written to the disk, and another subset is selected. During the writing operation, if a triple has been already added in a previous iteration, then its bitset is updated with the result of the logical *or* between the old bitset and the new bitset.

Bibliography

[Abel 2007] Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause and Daniel Olmedilla. *Enabling advanced and context-dependent access control in RDF stores.* In The Semantic Web, pages 1–14. Springer, 2007. (Cited on pages 7, 34, 35, 37, 38, 39, 41 and 80.)

[Abiteboul 1995] Serge Abiteboul, Richard Hull and Victor Vianu. Foundations of databases. Addison-Wesley, Boston, 1995. (Cited on page 21.)

[Adam 1989] Nabil R. Adam and John C. Wortmann. *Security-Control Methods for Statistical Databases: A Comparative Study.* ACM Comput. Surv., vol. 21, no. 4, pages 515–556, 1989. (Cited on page 66.)

[Agrawal 2004] Rakesh Agrawal, Roberto Bayardo, Christos Faloutsos, Jerry Kiernan, Ralf Rantzau and Ramakrishnan Srikant. *Auditing compliance with a hippocratic database.* In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pages 516–527. VLDB Endowment, 2004. (Cited on page 4.)

[Angles 2008] Renzo Angles and Claudio Gutiérrez. *Survey of graph database models.* ACM Comput. Surv., vol. 40, no. 1, 2008. (Cited on page 14.)

[Atre 2010] Medha Atre, Vineet Chaoji, Mohammed J. Zaki and James A. Hendler. *Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data.* In Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010, pages 41–50, 2010. (Cited on page 104.)

[Baldwin 1990] R. W. Baldwin. *Naming and Grouping Privileges to Simplify Security Management in Large Databases.* In Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990, pages 116–132, 1990. (Cited on page 29.)

[Barker 2002] Steve Barker. *Protecting deductive databases from unauthorized retrieval and update requests.* Data & Knowledge Engineering, vol. 43, no. 3, pages 293–315, 2002. (Cited on page 34.)

[Bell 1973] D Elliott Bell and Leonard J LaPadula. *Secure computer systems: Mathematical foundations*. Rapport technique, DTIC Document, 1973. (Cited on page 27.)

[Berners-Lee 2006] Tim Berners-Lee. *Linked data-design issues*. 2006. (Cited on page 2.)

[Berners-Lee 2011] Tim Berners-Lee and Dan Connolly. *Notation3 (N3): A readable RDF syntax*. W3C Team Submission, 2011. (Cited on page 18.)

[Bertino 1997] Elisa Bertino, Pierangela Samarati and Sushil Jajodia. *An Extended Authorization Model for Relational Databases*. IEEE Trans. Knowl. Data Eng., vol. 9, no. 1, pages 85–101, 1997. (Cited on page 32.)

[Bizer 2009] Christian Bizer, Tom Heath and Tim Berners-Lee. *Linked Data - The Story So Far*. Int. J. Semantic Web Inf. Syst., vol. 5, no. 3, pages 1–22, 2009. (Cited on page 1.)

[Broekstra 2003] Jeen Broekstra and Arjohn Kampman. *SeRQL: a second generation RDF query language*. In SWAD-Europe Workshop on Semantic Web Storage and Retrieval, pages 13–14, 2003. (Cited on page 18.)

[Costabello 2012] Luca Costabello, Serena Villata, Nicolas Delaforge, Fabien Gandon*et al.* *Linked data access goes mobile: Context-aware authorization for graph stores*. In LDOW-5th WWW Workshop on Linked Data on the Web-2012, 2012. (Cited on pages 34, 35, 37, 38, 39 and 80.)

[Cuppens 2003] Frédéric Cuppens and Alexandre Miège. *Modelling Contexts in the Or-BAC Model*. In 19th Annual Computer Security Applications Conference (ACSAC 2003), 8-12 December 2003, Las Vegas, NV, USA, pages 416–425, 2003. (Cited on page 26.)

[Cyganiak 2014] Richard Cyganiak, David Wood and Markus Lanthaler. *RDF 1.1 concepts and abstract syntax*. W3C Recommendation. Feb, 2014. (Cited on page 15.)

[Damiani 2002] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi and Pierangela Samarati. *A fine-grained access control system for XML documents*. ACM Transactions on Information and

System Security (TISSEC), vol. 5, no. 2, pages 169–202, 2002. (Cited on page 4.)

[Denning 1976] Dorothy E. Denning. *A Lattice Model of Secure Information Flow*. Commun. ACM, vol. 19, no. 5, pages 236–243, 1976. (Cited on page 27.)

[Diffie 1976] Whitfield Diffie and Martin E Hellman. *New directions in cryptography*. Information Theory, IEEE Transactions on, vol. 22, no. 6, pages 644–654, 1976. (Cited on page 4.)

[Farkas 2002] Csilla Farkas and Sushil Jajodia. *The Inference Problem: A Survey*. SIGKDD Explorations, vol. 4, no. 2, pages 6–11, 2002. (Cited on pages 6 and 66.)

[Farrell 2002] Stephen Farrell and Russell Housley. *An internet attribute certificate profile for authorization*. RFC3281, 2002. (Cited on page 83.)

[Ferraiolo 2001] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli. *Proposed NIST standard for role-based access control*. ACM Trans. Inf. Syst. Secur., vol. 4, no. 3, pages 224–274, 2001. (Cited on page 29.)

[Flouris 2010] Giorgos Flouris, Irini Fundulaki, Maria Michou and Grigoris Antoniou. *Controlling access to RDF graphs*. In Future Internet-FIS 2010, pages 107–117. Springer, 2010. (Cited on pages 7, 34, 35, 37, 38, 39, 41 and 84.)

[Griffiths 1976] Patricia P. Griffiths and Bradford W. Wade. *An Authorization Mechanism for a Relational Database System*. ACM Trans. Database Syst., vol. 1, no. 3, pages 242–255, 1976. (Cited on page 4.)

[Harris 2013] Steve Harris and Andy Seaborne. *SPARQL 1.1 query language*. W3C Recommendation, vol. 21, 2013. http://www.w3.org/TR/sparql11-query/. (Cited on page 18.)

[Haslhofer 2011] Bernhard Haslhofer, Elaheh Momeni Roochi, Bernhard Schandl and Stefan Zander. *Europeana RDF store report*. 2011. (Cited on page 11.)

[Hayes 2014] Patrick J Hayes and Peter F Patel-Schneider. *RDF 1.1 Semantics.* W3C Recommendation, 2014. http://www.w3.org/TR/rdf11-mt/. (Cited on pages 7, 14, 16, 17 and 25.)

[Jain 2006] Amit Jain and Csilla Farkas. *Secure resource description framework: an access control model.* In SACMAT, pages 121–129. ACM, 2006. (Cited on pages 7, 9, 34, 35, 37, 38, 39, 40 and 67.)

[Kahn 1962] A. B. Kahn. *Topological sorting of large networks.* Commun. ACM, vol. 5, no. 11, pages 558–562, 1962. (Cited on page 60.)

[Kobilarov 2009] Georgi Kobilarov, Tom Scott, Yves Raimond, Silver Oliver, Chris Sizemore, Michael Smethurst, Christian Bizer and Robert Lee. *Media Meets Semantic Web - How the BBC Uses DBpedia and Linked Data to Make Connections.* In The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings, pages 723–737, 2009. (Cited on page 2.)

[Kolovski 2010] Vladimir Kolovski, Zhe Wu and George Eadon. *Optimizing Enterprise-Scale OWL 2 RL Reasoning in a Relational Database System.* In The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I, pages 436–452, 2010. (Cited on page 77.)

[Lamport 1981] Leslie Lamport. *Password authentication with insecure communication.* Communications of the ACM, vol. 24, no. 11, pages 770–772, 1981. (Cited on page 4.)

[Lampson 1974] Butler W. Lampson. *Protection.* Operating Systems Review, vol. 8, no. 1, pages 18–24, 1974. (Cited on page 26.)

[Lassner 2002] David Lassner, Dave De Roure and Arun Iyengar, editeurs. Proceedings of the eleventh international world wide web conference, WWW 2002, may 7-11, 2002, honolulu, hawaii. ACM, 2002. (Cited on page 18.)

[Livshits 2005] V. Benjamin Livshits and Monica S. Lam. *Finding Security Vulnerabilities in Java Applications with Static Analysis.* In Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005, 2005. (Cited on page 68.)

[Lopes 2010] Nuno Lopes, Axel Polleres, Umberto Straccia and Antoine Zimmermann. *AnQL: SPARQLing up annotated RDFS*. In The Semantic Web–ISWC 2010, pages 518–533. Springer, 2010. (Cited on pages 35, 38 and 95.)

[Lopes 2012] Nuno Lopes, Sabrina Kirrane, Antoine Zimmermann, Axel Polleres and Alessandra Mileo. *A Logic Programming approach for Access Control over RDF*. In ICLP 2012, Hungary, pages 381–392, 2012. (Cited on pages 7, 9, 34, 37, 38, 39, 40, 41 and 80.)

[Martelli 1982] Alberto Martelli and Ugo Montanari. *An Efficient Unification Algorithm*. ACM Trans. Program. Lang. Syst., vol. 4, pages 258–282, April 1982. (Cited on pages 70, 122 and 123.)

[McLean 1990] John McLean. *Security Models and Information Flow*. In Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990, pages 180–189, 1990. (Cited on page 111.)

[Papakonstantinou 2012] Vassilis Papakonstantinou, Maria Michou, Irini Fundulaki, Giorgos Flouris and Grigoris Antoniou. *Access control for RDF graphs using abstract models*. In SACMAT, pages 103–112, 2012. (Cited on pages 7, 9, 35, 37, 38, 39, 40 and 94.)

[Pérez 2009] Jorge Pérez, Marcelo Arenas and Claudio Gutierrez. *Semantics and complexity of SPARQL*. ACM Trans. Database Syst., vol. 34, no. 3, pages 16:1–16:45, September 2009. (Cited on pages 16 and 19.)

[Polleres 2007] Axel Polleres. *From SPARQL to Rules (and Back)*. In WWW, pages 787–796, 2007. (Cited on pages 18 and 34.)

[Prud'hommeaux 2014] Eric Prud'hommeaux, José Emilio Labra Gayo and Harold R. Solbrig. *Shape expressions: an RDF validation and transformation language*. In Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014, pages 32–40, 2014. (Cited on page 111.)

[Rabitti 1991] Fausto Rabitti, Elisa Bertino, Won Kim and Darrell Woelk. *A Model of Authorization for Next-Generation Database Systems*. ACM Trans. Database Syst., vol. 16, no. 1, pages 88–131, 1991. (Cited on page 4.)

[Rachapalli 2014] Jyothsna Rachapalli, Vaibhav Khadilkar, Murat Kantarcioglu and Bhavani Thuraisingham. *Towards Fine Grained RDF Access Control*. In SACMAT, pages 165–176. ACM, 2014. (Cited on page 7.)

[Ramli 2011] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson and Flemming Nielson. *The Logic of XACML*. In FACS, 2011, Norway, pages 205–222, 2011. (Cited on page 30.)

[Reddivari 2005] Pavan Reddivari, Tim Finin and Anupam Joshi. *Policy-based access control for an RDF store*. In WWW, pages 78–81, 2005. (Cited on pages 7, 9, 34, 35, 37, 38, 39, 40, 41 and 80.)

[Rivest 1978] Ronald L Rivest, Adi Shamir and Len Adleman. *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM, vol. 21, no. 2, pages 120–126, 1978. (Cited on page 4.)

[Samarati 2001] Pierangela Samarati and Sabrina Capitani de Vimercati. *Access control: Policies, models, and mechanisms*. In Foundations of Security Analysis and Design, pages 137–196. Springer, 2001. (Cited on pages 25 and 32.)

[Sandhu 1996] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein and Charles E. Youman. *Role-Based Access Control Models*. IEEE Computer, vol. 29, no. 2, pages 38–47, 1996. (Cited on page 29.)

[Sandhu 2000] Ravi S. Sandhu, David F. Ferraiolo and D. Richard Kuhn. *The NIST model for role-based access control: towards a unified standard*. In ACM Workshop on Role-Based Access Control, pages 47–63, 2000. (Cited on page 29.)

[Seaborne 2004] Andy Seaborne. *RDQL-a query language for RDF*. W3C Member submission, vol. 9, no. 29-21, page 33, 2004. (Cited on page 18.)

[Sintek 2002] Michael Sintek and Stefan Decker. *TRIPLE—A query, inference, and transformation language for the semantic web*. In The Semantic Web—ISWC 2002, pages 364–378. Springer, 2002. (Cited on page 18.)

[Su 1987] Tzong-An Su and Gultekin Özsoyoglu. *Data Dependencies and Inference Control in Multilevel Relational Database Systems*. In Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 27-29, 1987, pages 202–211, 1987. (Cited on page 66.)

[ter Horst 2005] Herman J. ter Horst. *Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*. J. Web Sem., vol. 3, no. 2-3, pages 79–115, 2005. (Cited on page 18.)

[Thuraisingham 1987] Bhavani M. Thuraisingham. *Security checking in relational database management systems augmented with inference engines*. Computers & Security, vol. 6, no. 6, pages 479–492, 1987. (Cited on page 66.)

[Udrea 2010] Octavian Udrea, Diego Reforgiato Recupero and V. S. Subrahmanian. *Annotated RDF*. ACM Trans. Comput. Log., vol. 11, no. 2, 2010. (Cited on page 34.)

[Wang 2004] Lingyu Wang, Duminda Wijesekera and Sushil Jajodia. *A logic-based framework for attribute based access control*. In ACM Workshop, pages 45–55, 2004. (Cited on page 26.)

Appendix A

Appendix

In this appendix, we provide additional technical definitions and lemmas and the LUBM Queries used for experiments.

A.1 Results from Section 2.1.1.3

Lemma A.1.1 (Closure property) Cl *is a closure operator that satisfies the following properties for all* $G \in \mathsf{BGP}$ *:*

Extensive $G \subseteq \mathsf{Cl}(G)$

Increasing $G \subseteq H \Rightarrow \mathsf{Cl}(G) \subseteq \mathsf{Cl}(H)$

Idempotent $\mathsf{Cl}(G) = \mathsf{Cl}(\mathsf{Cl}(G))$

Proof of Lemma A.1.1 First, let k the smallest natural number that exists according to Lemma 2.1.14, so we can express $\mathsf{Cl}(G)$ as a finite union of graphs $\mathsf{Cl}(G) = G_0 \cup G_1 \cup \ldots \cup G_k$.

Extensive By definition $G = G_0$.

Increasing By induction, we show first that $G_i \subseteq H_i$. For $i = 0$, the result holds by hypothesis $G_0 = G \subseteq H = H_0$. Assume that $G_i \subseteq H_i$ holds, we have $G_{i+1} = G_i \cup \phi(G_i)$. Consider $x \in G_{i+1}$, if $x \in G_i$ then $x \in H_i$ and we're done by the induction hypothesis. Otherwise, it is the case that $x \in \phi(G_i)$ and there exists a rule r such that $x \in \phi_r(G_i)$, but r can be applied on H_i as well because $G_i \subseteq H_i$, so $x \in \phi_r(H_i) \subseteq H_{i+1}$ and we're done again. There exists some $l \in \mathbb{N}$ such that $\mathsf{Cl}(H) = H_0 \cup H_1 \cup \ldots \cup H_l$ with $k \leq l$, by the previous result $G_i \subseteq H_i$ hold for all $i \in 0 \ldots k$ thus $\mathsf{Cl}(G) \subseteq \mathsf{Cl}(H)$ by piecewise union.

Idempotent By Lemma 2.1.14 we can write $\mathsf{Cl}(\mathsf{Cl}(G)) = \mathsf{Cl}(G)_0 \cup \mathsf{Cl}(G)_1 \cup \ldots \cup \mathsf{Cl}(G)_{k'}$ for some k', we have to show that $\mathsf{Cl}(G)_i = \mathsf{Cl}(G)_0$ so the union collapses at $\mathsf{Cl}(G)_0 = \mathsf{Cl}(G)$. The proof amounts to show that $\mathsf{Cl}(G) = \mathsf{Cl}(G) \cup \phi(\mathsf{Cl}(G))$, or equivalently that $\phi(\mathsf{Cl}(G)) \subseteq \mathsf{Cl}(G)$. For the sake of the contradiction, assume that there is some $x \in \phi(\mathsf{Cl}(G))$ with $x \notin \mathsf{Cl}(G)$, by Definition 2.1.13, there is a rule $r = (tp \leftarrow tp_1, \ldots, tp_n)$ and a mapping $\sigma \in [\![\{tp_1, \ldots, tp_n\}]\!]_{\mathsf{Cl}(G)}$ such that $x = (tp)\sigma$. Because $(\{tp_1, \ldots, tp_n\})\sigma \subseteq \mathsf{Cl}(G)$ from Definition 2.1.8, we can can build a function $\iota : \{1 \ldots n\} \rightarrow \{1 \ldots k\}$ that maps the index i of each tp_i to the index j of some subgraph G_j of $\mathsf{Cl}(G)$. Let m be the largest natural number in the set $\{\iota(1) \ldots \iota(n)\}$. If $m \neq k$, then $x \in \phi(G_m) \subseteq \mathsf{Cl}(G)$, contradicting $x \notin \mathsf{Cl}(G)$. Otherwise, $m = k$ so $G_{k+1} \neq G_k$, and k is not the smallest natural number that stabilizes $\mathsf{Cl}(G)$ according to Lemma 2.1.14, a contradiction again.

A.2 Results from Section 4.3

A.2.1 Additional Definitions

Definition A.2.1 (Renaming Substitution) *A* renaming substitution *for a basic graph pattern B is a substitution $\rho : \mathsf{var}(B) \rightarrow \mathsf{V}$ associating each variable of B to a fresh variable. A* renaming substitution *for an authorization $\mathpzc{a}$ is a renaming substitution for* $\mathsf{hb}(\mathpzc{a})$*.*

Definition A.2.2 (Unifier, Most General Unifier [Martelli 1982]) *A substitution μ is a* unifier *for two triple patterns tp_1 and tp_2 if $tp_1\mu = tp_2\mu$. A substitution μ is a unifier for two* tuples *of triple patterns $TP = (tp_1, \ldots, tp_k)$ and $TP' = (tp'_1, \ldots, tp'_k)$ of the same size k, if for all $i \in \{1, \ldots, k\}$, μ is a unifier for tp_i and tp'_i.*

Two triples patterns tp_1 and tp_2 (resp. two tuples of triple patterns TP and TP') are unifiable *if there exists a unifier μ for tp_1 and tp_2 (resp. TP and TP').*

A substitution μ is a most general unifier *for two triple patterns tp_1 and tp_2 (resp. two tuples of triple patterns TP and TP') if for any unifier μ' of tp_1*

and tp_2 (resp. TP and TP'), there exists a substitution η such that $\mu' = \mu\eta$.

Lemma A.2.3 (Computable Most General Unifier [Martelli 1982]) *If TP and TP' are unifiable, then there exists* a most general unifier μ *for TP and TP'. Given two unifiable tuples of triple patterns TP and TP', the function* mgu(TP, TP') *returns one of the most general unifiers of TP and TP'. Moreover, this most general unifier is unique up to a renaming of variables.*

A.3 LUBM Queries

- **Q1**: `SELECT ?x ?y ?z WHERE { ?z ub:subOrganizationOf ?y. ?y rdf:type ub:University. ?z rdf:type ub:Department. ?x ub:memberOf ?z. ?x rdf:type ub:GraduateStudent. ?x ub:undergraduateDegreeFrom ?y. }`

- **Q2**: `SELECT ?x WHERE { ?x rdf:type ub:Course. ?x ub:name ?y. }`

- **Q3**: `SELECT ?x ?y ?z WHERE { ?x rdf:type ub:UndergraduateStudent. ?y rdf:type ub:University. ?z rdf:type ub:Department. ?x ub:memberOf ?z. ?z ub:subOrganizationOf ?y. ?x ub:undergraduateDegreeFrom ?y. }`

- **Q4**: `SELECT ?x WHERE { ?x ub:worksFor <http://www.Department0.University0.edu>. ?x rdf:type ub:FullProfessor. ?x ub:name ?y1. ?x ub:emailAddress ?y2. ?x ub:telephone ?y3. }`

- **Q5**: `SELECT ?x WHERE { ?x ub:subOrganizationOf <http://www.Department0.University0.edu>. ?x rdf:type` ub:ResearchGroup }

- **Q6**: `SELECT ?x ?y WHERE { ?y ub:subOrganizationOf <http://www.University0.edu>. ?y rdf:type ub:Department. ?x` ub:worksFor ?y. `?x rdf:type ub:FullProfessor. }`

- **Q7**: `SELECT ?x ?y ?z WHERE { ?y ub:teacherOf ?z. ?y rdf:type ub:FullProfessor. ?z rdf:type ub:Course. ?x ub:advisor ?y. ?x rdf:type ub:UndergraduateStudent. ?x ub:takesCourse ?z }`