TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Theoretical Computer Science, Chair of Automata Theory

Martin Knechtel

Access Restrictions to and with Description Logic Web Ontologies

Access Restrictions to and with Description Logic Web Ontologies

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Ing. Martin Knechtel
geboren am 21. Juli 1982 in Dresden

verteidigt am 10. Dezember 2010

Gutachter:

Prof. Dr.-Ing. Franz Baader,
Technische Universität Dresden

Prof. Dr. Heiner Stuckenschmidt,
Universität Mannheim

Dresden im Dezember 2010

To those who
inspired me.

Abstract

Access restrictions are essential in standard information systems and became an issue for ontologies in the following two aspects. Ontologies can represent explicit and implicit knowledge about an access policy. For this aspect we provided a methodology to represent and systematically complete role-based access control policies. Orthogonally, an ontology might be available for limited reading access. Independently of a specific ontology language or reasoner, we provided a lattice-based framework to assign labels to an ontology's axioms and consequences. We looked at the problems to compute and repair one or multiple consequence labels and to assign a query-based access restriction. An empirical evaluation has shown that the algorithms perform well in practical scenarios with large-scale ontologies.

Contents

List of Algorithms

List of Figures

List of Tables

List of Acronyms

cMCS	candidate MCS	96
CS	change set	78
DAC	discretionary access control	32
DL	Description Logic	1
FCA	Formal Concept Analysis	11
FP	full axiom pinpointing	104
GCI	general concept inclusion	21
HST	Hitting Set Tree	25
IAS	minimal inserted axiom set	79
KR	knowledge representation	2
LBAC	lattice-based access control	17
LP	label-optimized axiom pinpointing	105
MCS	minimal multiple change set	95
MinA	minimal axiom set	24
MinCS	minimal CS	78
MinLab	minimal label set	69
OWL	Web Ontology Language	2
RAS	minimal removed axiom set	79
RBAC	role-based access control	1
RBAC-CH	RBAC with object class hierarchy	38
RMCS	relaxed MCS	99
W3C	World Wide Web Consortium	2

1 Introduction

Access control is an essential operation in standard information systems to prevent unauthorized access and use of information. As semantic technology is more and more applied in real-world applications, access control also becomes an issue for ontologies in the following two ways. Ontologies can, on the one hand, be a great help in representing knowledge about an access policy, e.g. including the definition of user roles based on the attributes of persons. An ontology might, on the other hand, be an object of a computing system which is available for reading access under certain restrictions only. Deciding what parts of an ontology can be provided to the requesting person becomes much harder compared to systems where all the available information is stored explicitly, since additional knowledge can be inferred by reasoning.

In this thesis, the twofold application of access restrictions and knowledge representation with Description Logic (DL) Web ontologies have been investigated, on the one hand to represent knowledge about role-based access control (RBAC) policies and on the other hand to represent knowledge under access restrictions. The results include:

- a methodology to create a representation of an RBAC policy in Description Logics and to complete it in a computer-supported dialog with the security engineer in a systematic and non-redundant procedure;
- a framework, which is independent of a specific ontology language or reasoner, that allows to restrict reading access to explicit knowledge represented by an ontology and implicit knowledge that follows from this ontology;
- user support for a security engineer to facilitate the task of assigning access restrictions to individual axioms by grouping them document-based or query-based in an intuitive manner;
- an empirical evaluation, showing that concepts and algorithms developed in this PhD project perform well in practical scenarios with large-scale ontologies.

1.1 Access Restrictions to and with Description Logic Web Ontologies

Description Logics [Baa+07b] have been successfully used to represent knowledge for a wide variety of real-world application domains. The most popular application is the Semantic Web. The World Wide Web Consortium (W3C) developed and recommended the Web Ontology Language (OWL) as the standard ontology language for the Semantic Web [MPSP09]. DLs provide the basis of OWL.

The relevant portions of a considered domain are described through a DL *ontology*. Rather than having an ontology explicitly stating every piece of knowledge, one would prefer to be able to infer additional knowledge that appears implicitly in this ontology. For example, knowing that

- Martin's camera is a digital single-lens reflex camera and knowing also that
- every digital single-lens reflex camera is a single-lens reflex camera,

it should be unnecessary to express that

- Martin's camera is a single-lens reflex camera

because this is a direct *consequence* following from the other two pieces of knowledge which we call *axioms*. For each consequence, one or more *explanations* can be given in the form of minimal sets of axioms that imply the consequence. The available highly-optimized DL reasoners compute consequences from axioms. In other words, they make the *implicit* knowledge in the ontology *explicit*. The semantics of an ontology language determines which consequences can be derived. DLs have a declarative semantics, i.e. it depends only on the represented knowledge, which ensures that different implementations of reasoning services lead to the same results. This was not guaranteed with early knowledge representation (KR) systems that had an operational semantics, i.e. defined by means of the procedures that perform the reasoning tasks.

Two main aspects appear with respect to access restrictions and ontologies, which are both covered by this thesis. These are

- restricting reading access to ontologies, and
- representing access restrictions with ontologies.

Especially the first aspect recently earned interest. Fine-grained access control has been identified in [Bru+07] as an "important issue to meet business requirements" that will "affect storage, query and reasoning systems" and hence is a "key issue to explore." This thesis provides a framework to restrict reading access to an ontology in a fine-grained manner. The general idea of the framework is to assign a *label* to each axiom which represents an access restriction and to be able to compute such a label also for every consequence. The general idea also includes being able to change axiom labels systematically in order to obtain intended consequence labels.

For the second aspect of representing access restrictions with ontologies, we focus on RBAC [SFK00] policies in this thesis and show how to represent knowledge from

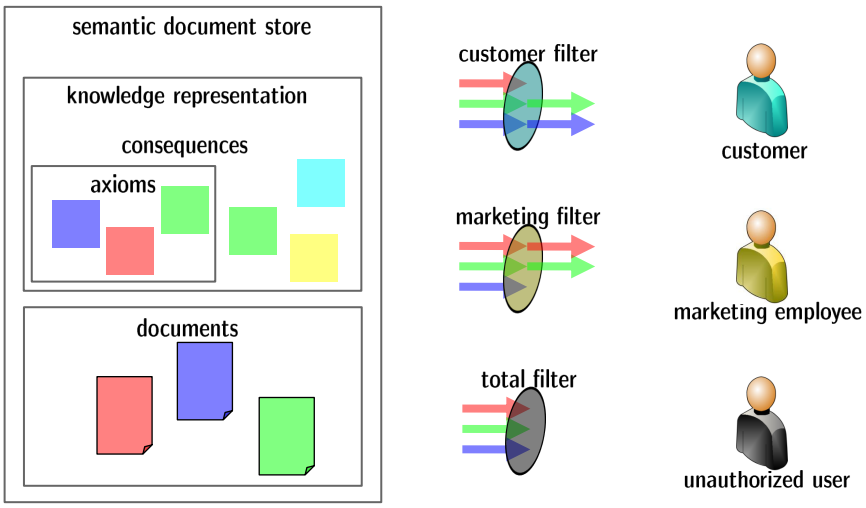

Figure 1.1: Access restricted semantic document store

such a policy with an ontology. This makes sense, as tasks related to RBAC policies can be reduced to standard DL reasoning. Furthermore, we show how to systematically complete this knowledge.

Both, the first and the second aspect, are represented in the following model for systems managing documents and ontologies with access restrictions. This model is called access restricted semantic document store. A typical instance of that model is a Semantic Web system with access restrictions. Figure 1.1 depicts the main components. It consists of storage for documents, storage for knowledge representation and filters to restrict access on both. The explicit axioms and implicit consequences represent knowledge from the documents. The set of axioms is contained in the set of consequences, since every axiom trivially follows from the set of axioms. The color of each user in the right part of Figure 1.1 represents the set of assigned user roles. The color of the documents represents the set of user roles allowed to *read* the document. Similarly, the color of the axioms and consequences represents the set of user roles allowed to read the knowledge representation. Intuitively, the access restrictions that apply for the documents should also apply for the axioms and their consequences.

In order to discuss technical concepts and results of the thesis by means of a realistic practical scenario, the following section introduces an instance of an access restricted semantic document store. This access restricted semantic document store manages documents and knowledge about products and allows controlled access by several types of users. The terminology which is used informally in the following section is defined formally later in Chapter 2.

1.2 Case Study: Access Restricted Semantic Product Documentation

The specific example scenario introduced here is part of the research project THESEUS/PROCESSUS [THE10]. Within this project, semantically annotated documents describe services offered and sold on a marketplace in the Web, like traditional goods would be sold on Amazon, eBay and similar Web marketplaces. Different types of users are involved with different permissions that allow them to create, advertise, sell, buy, etc. the services. Documents and knowledge are managed with an access restricted semantic document store, so that users can read and change the documents and query the knowledge representation according to the user's permissions. The service documents are semantically annotated using a vocabulary for service descriptions [Obe+09] and service documents [HBPK09]. In this scenario the documents are stored in a semantic Wiki which allows storing documents and annotations at one place. For each document, the annotations can be exported as a document ontology. For example the source text of a Semantic MediaWiki[1] article about a service called *ecoCalculatorV1* could contain the character sequence [[Category:EUecoService]] [[Category:HighperformanceService]] to classify the service as *EUecoService* and as *HighperformanceService*. The document ontology created by the OWL export of the Semantic MediaWiki would then contain the DL axiom[2] $EUecoService \sqcap HighperformanceService(ecoCalculatorV1)$. Alternatively, a document ontology could also be created manually by an ontology engineer after reading a document.

An approach to keep the information and the knowledge about a service in the software models, the source code, the documents, and the document ontologies consistent has been presented in [HBPK08; HBPK09] during this PhD project. Its details are out of scope of this thesis. However, it underlines the practical relevance of knowledge about software documents that is represented in a document ontology.

The union of all document ontologies and imported ontologies forms a large ontology which can be used to infer implicit consequences with a DL reasoner and to answer user queries. This scenario implies that the ontology contains knowledge which is as sensitive as the source documents. To make the sensitivity level of each axiom of the large ontology explicit, a label is assigned to each axiom. A similar label should be computable for each consequence.

The document types contained in our example are user manual, marketing document, customer contract document, terms of use document, installation guide, external technical interface document, design document and rating entry, abbreviated by UM, MD, CCD, ToUD, IG, ETID, DD, RE. Those document types are accessible by users with different user roles. The roles contained in our example are marketplace visitor, customer, development engineer, service vendor, legal department employee, service provider, marketing employee, technical editor and customer service employee, abbreviated by MV, CU, DE, SV, LDE, SP, ME, TE and CSE. Similar document types and target groups can be found in product standards of big software companies. An access control system reg-

[1] `http://semantic-mediawiki.org`

[2] The syntax and semantics of DL axioms is introduced later in detail.

	mayRead								mayWrite								mayApprove							
	UM	MD	CCD	ToUD	IG	ETID	DD	RE	UM	MD	CCD	ToUD	IG	ETID	DD	RE	UM	MD	CCD	ToUD	IG	ETID	DD	RE
MV		×		×				×																
CU	×	×	×	×		×		×								×			×	×				
DE	×	×		×	×	×	×	×	×				×	×	×			×						
SV	×	×	×	×	×	×	×	×									×	×	×	×	×	×	×	
LDE	×	×	×	×	×	×	×	×			×	×												
SP	×	×		×	×	×		×																
ME	×	×		×	×	×	×	×		×														
TE	×	×		×	×	×	×	×	×				×	×	×									
CSE	×	×	×	×	×	×	×	×			×													

Table 1.1: Our example RBAC matrix

ulates the allowed permissions for each user according to her user role. The actions in our example are mayApprove, mayWrite and mayRead, abbreviated by MA, MW and MR. The allowed actions of user roles on document types are defined by the *RBAC matrix* in Table 1.1. For example, a marketing document is intended to be publicly available so that, e.g., a marketplace visitor can read it. This is indicated by a cross at the intersection of the respective row and column. But a marketplace visitor is not allowed to read a design document, indicated[3] by a missing cross. Note that the defined allowed actions abstract from further constraints. Obviously a customer has access only to a customer contract document of her own purchased service, a development engineer has access only to a design document of a software she is working on, etc.

Figure 1.2 provides a set of user roles and a relation between user roles, expressing that one user role inherits all permissions from another user role in the direction the arrow is pointing. For example, a customer is allowed to do anything a marketplace visitor is allowed to do. As can be easily verified, this permission inheritance relation is consistent with the RBAC matrix in Table 1.1. The permission inheritance relation is an order on the set of user roles and is called user *role hierarchy* in an RBAC model. In RBAC, a user role hierarchy adds further implicit permissions to the initial set of explicit permissions. An RBAC matrix is a static representation of an RBAC policy, and can for example be used for efficient access on explicit and implicit permissions.

The set of user roles and their user role hierarchy may be represented by a type of partially ordered set, called a *lattice* $(L, \leqslant)$. For example, a customer is allowed to do anything a marketplace visitor is allowed to do, so $\mathsf{CU}, \mathsf{MV} \in L$ and $\mathsf{CU} \leqslant \mathsf{MV}$. The lattice can be computed from the RBAC matrix, as explained later. The lattice can be applied in an access control model that allows a user labeled ℓ_u to read an axiom labeled ℓ_a iff $\ell_u \leqslant \ell_a$. In such a model, an axiom label intuitively represents a boundary dividing the set of user labels into those with and those without reading permission. For example, an ontology axiom labeled with CU is readable by any user

[3]In fact, this is only one possible reading of the matrix and others are discussed later. For the time being only this reading is used.

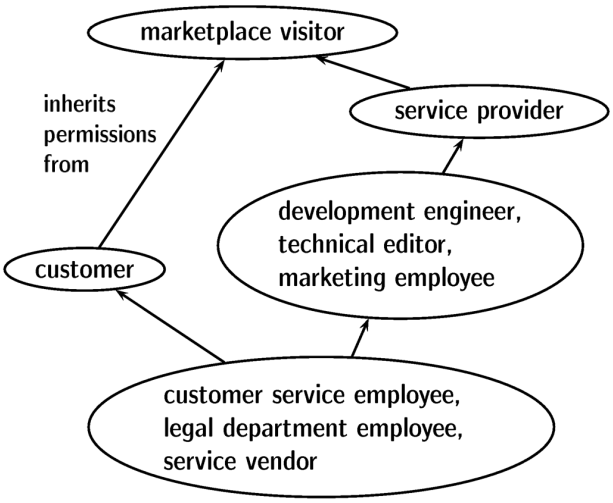

Figure 1.2: User roles with permission inheritance relation

labeled CU, CSE, LDE or SV. The following example contains an ontology O with labeled axioms, where each axiom came from a different document ontology.

Example 1. Let $(L, \leqslant)$ be the lattice shown in Figure 1.3, where elements $\ell_0, \ell_2, \ell_3, \ell_5$ represent user roles as illustrated. Moreover, let O be a labeled ontology from a marketplace in the Semantic Web with the following five axioms

$$\begin{aligned}
&a_1 : \mathit{EUecoService} \sqcap \mathit{HighperformanceService}(\mathit{ecoCalculatorV1})\\
&a_2 : \mathit{HighperformanceService}\\
&\quad \sqsubseteq \mathit{ServiceWithLowCustomerNr} \sqcap \mathit{LowProfitService}\\
&a_3 : \mathit{EUecoService} \sqsubseteq \mathit{ServiceWithLowCustomerNr} \sqcap \mathit{LowProfitService}\\
&a_4 : \mathit{ServiceWithLowCustomerNr} \sqsubseteq \mathit{ServiceWithComingPriceIncrease}\\
&a_5 : \mathit{LowProfitService} \sqsubseteq \mathit{ServiceWithComingPriceIncrease}
\end{aligned}$$

where the function lab assigns to each axiom a_i the label ℓ_i, which is indicated also in Figure 1.3. The consequence

$$c_1 : \mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$$

follows from each of the explanations $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}, \{a_1, a_3, a_5\}$. The consequence

$$c_2 : \mathit{LowProfitService}(\mathit{ecoCalculatorV1})$$

follows from each of the explanations $\{a_1, a_2\}, \{a_1, a_3\}$. Additionally, the concept assertions of *ecoCalculatorV1* to the concepts *EUecoService*, *HighperformanceService* and *ServiceWithLowCustomerNr* are consequences of O.

1.3 Requirements

User stories have been introduced for agile software projects to describe requirements shortly and informally in few sentences [Coh04] from a user perspective. To define the

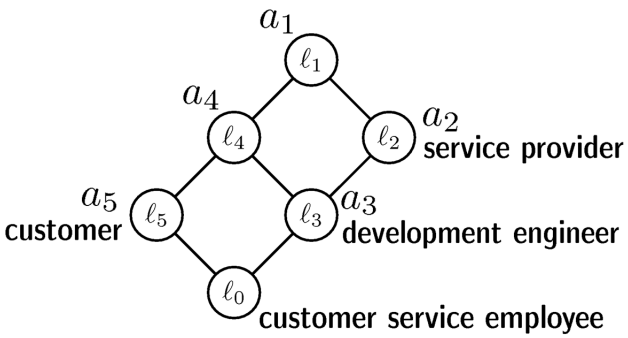

Figure 1.3: Lattice with 4 user labels and an assignment of 5 axioms to labels

requirements for the scenario presented in the previous section, this section provides the following user stories. The diagram in Figure 1.4 uses UML use case notation and shows which actor is involved in which user story by a solid line. It also shows which user story extends another one by an arrow pointing to the extended user story.

User Story 1 (Define visible sub-ontology)**.** The security administrator defines the visible sub-ontology for each user by means of labeled axioms.

User Story 2 (Retrieve visible sub-ontology)**.** The user retrieves her visible sub-ontology based on her user label.

User Story 3 (Compute label of a consequence)**.** The security administrator is interested in the label of an inferred consequence. Intuitively, this should be based on the labels of the explicit axioms.

User Story 4 (Retrieve visible consequences)**.** The user wants to retrieve the set of consequences which follow from her visible sub-ontology, i.e. the set of consequences she could compute on her own. As explained before, the set of consequences includes also the set of axioms.

User Story 5 (Distribute visible consequences as a view)**.** Different users, which could be be also groups of users, user roles, organizations, or organizational units, want to retrieve their visible consequences. This allows reuse of explicit and implicit knowledge across security domains as illustrated in Figure 1.5. For example, knowledge from flyers and other marketing documents is distributed to the company Web site and a marketplace portal whereas knowledge from design documents and test protocols remains at the team portal.

User Story 6 (User dependent query answering)**.** Users posing queries against the system retrieve answers with respect to their user roles, as illustrated in Figure 1.6.

User Story 7 (Repair label of a consequence)**.** The security administrator is not satisfied with a consequence's label. Since it can only be changed indirectly by changing the axiom's labels, she requests a proposed minimal axiom re-labeling.

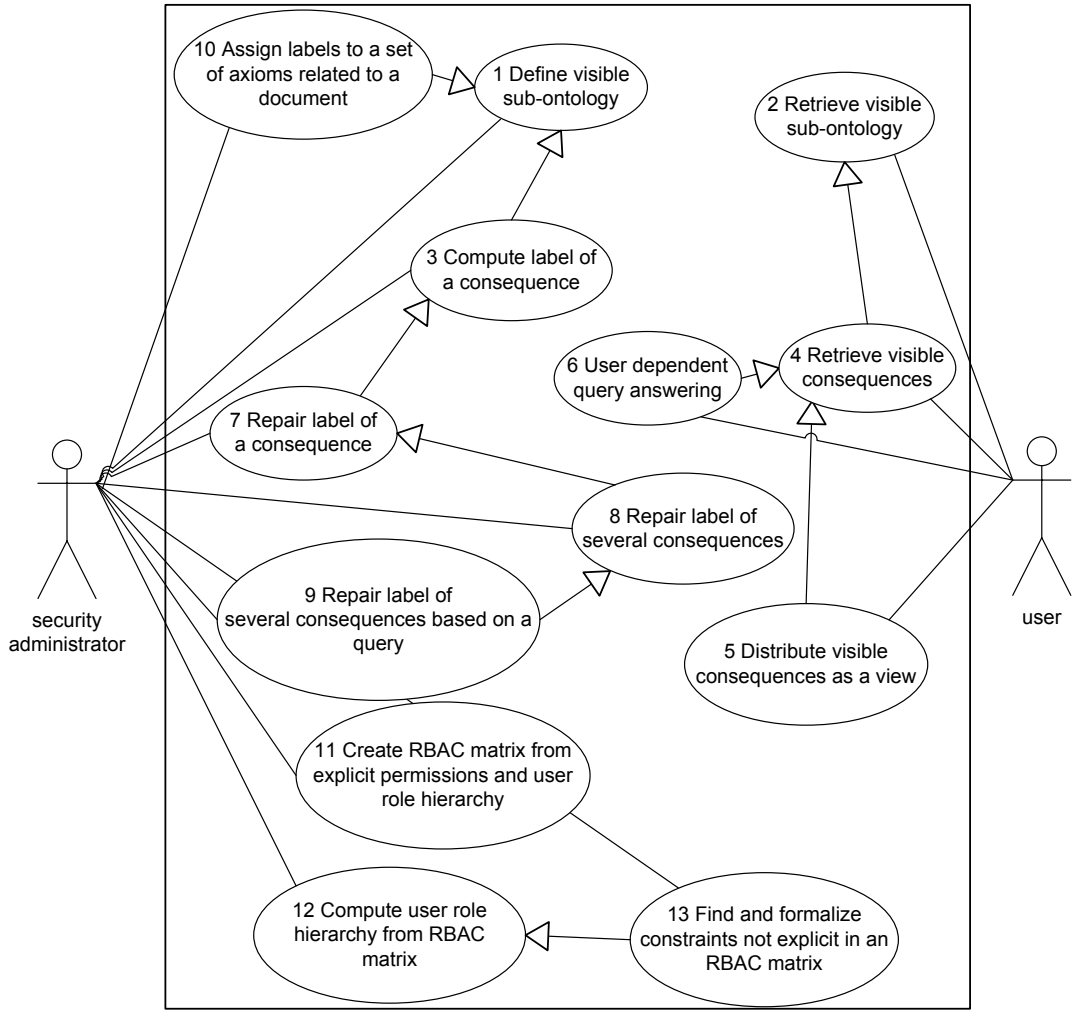

Figure 1.4: User stories

User Story 8 (Repair label of several consequences)**.** Given not only one consequence, but a set of consequences and their respectively intended consequence label, the security administrator requests a proposed minimal axiom re-labeling.

User Story 9 (Repair label of several consequences based on a query)**.** Instead of enumerating a set of consequences with their intended label, the security administrator defines this set intentionally by a query. The query could, e.g., address consequences about a concept and all subconcepts in order to restrict knowledge along the subsumption hierarchy similar to restricting access to files in a directory and all subdirectories.

User Story 10 (Assign labels to a set of axioms related to a document)**.** Instead of assigning labels to single axioms individually, the security administrator wants to address a set of axioms. She wants to transfer the access restrictions that apply to a document also to all axioms of the respective document ontology. Especially for a

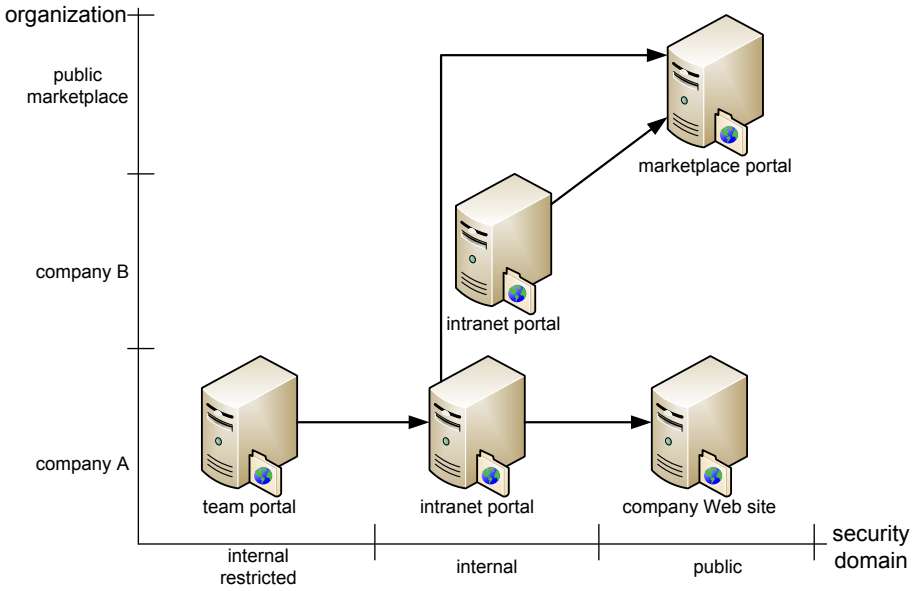

Figure 1.5: Distribution of parts of explicit and implicit product knowledge across security domains

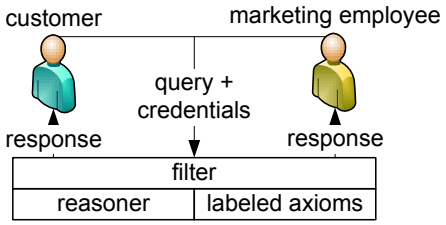

Figure 1.6: User dependent responses to one and the same query

security administrator who is familiar with access restrictions to a file system but not (yet) to ontologies, this might increase usability.

User Story 11 (Create RBAC matrix from explicit permissions and user role hierarchy)**.** Given a set of explicit permissions and a user role hierarchy, the security administrator might be interested in computing a complete RBAC matrix containing the union of explicit and implicit permissions.

User Story 12 (Compute user role hierarchy from RBAC matrix)**.** Given the union of explicit and implicit permissions in the form of an RBAC matrix, for example from a legacy system, the security administrator is interested in a user role hierarchy that is consistent with the RBAC matrix. This helps her to create a clearer policy with fewer explicit permissions while keeping the union of explicit and implicit permissions. Furthermore, she wants to use it as a basis for the labeling lattice in the user stories given above.

User Story 13 (Find and formalize constraints not explicit in an RBAC matrix)**.** Given the union of explicit and implicit permissions in the form of an RBAC matrix, the system administrator is interested to find not only the user role hierarchy, but further general constraints. For example, the constraint "no user is allowed to write and approve the same document" is true in the RBAC matrix in Table 1.1, but only the security administrator can decide if it is true in general. She is interested in a computer-supported interview to find all constraints systematically and with no redundant questions.

1.4 Research Questions

Based on the user stories above, this section lists the identified research questions. The User Stories 1, 2, 3, 4, 5 and 6 can be made technically possible by answering the following research questions.

1. For an access control model which allows multiple visible sub-ontologies from one large ontology with respect to a criterion expressed by axiom labels, user labels and a hierarchy between those labels, how can such a model be formalized and what constraints need to be taken into account?

2. Given that all axioms of an ontology are labeled, what is the label of a consequence which can be inferred and made explicit by a reasoner?

Repairing one or several consequence labels as described in User Stories 7 and 8 and the usability issues in User Stories 9 and 10 require an answer to the following research questions.

3. Can the repair of a consequence label be reduced to a known problem, and what optimizations are possible?

4. How can repair of a consequence label be extended to the repair of multiple consequence labels in parallel?

5. How are conflicts handled that might appear when goal labels of multiple consequences interfere with each other?

6. Is it possible to increase usability by reusing document access restrictions for document ontologies or by assigning a label to a set of consequences intentionally defined by a query?

In order to make User Stories 11, 12 and 13 technically possible, the following research questions need to be solved.

7. Are DLs an appropriate formal representation of the explicit permissions defined by an RBAC policy and is it possible to infer the implicit permissions from this representation?

8. Can a labeling lattice be obtained from an existing RBAC matrix and what may need to be changed?

9. Can the systematic and non-redundant identification of constraints in an RBAC matrix be reduced to a known problem and what extensions may be necessary?

The following section provides a detailed architecture of an access restricted semantic document store with the required components. It also provides an overview on the chapters of this thesis and their relation to architecture components, research questions and relevant publications created during this PhD project.

1.5 Architecture and Dissertation Outline

The overall architecture of an access restricted semantic document store is depicted in Figure 1.7. Circles represent components, rectangles represent data, and each arrow represents a flow of data. The blue blocks indicate which parts of the architecture are covered by topics in the chapters of this thesis. In the following we give an overview of the architecture and the topics covered in the chapters. Components and data of the architecture introduced in the text are written *emphasized*.

Chapter 2 introduces technical preliminaries to make this thesis' technical concepts and results accessible. It covers knowledge representations, lattices and Formal Concept Analysis (FCA), access control and inference control.

In the beginning of this PhD project, initial ideas have been published at PhD Symposia.

[Kne08a] M. Knechtel. "Access restriction inside ontologies." In: *Proceedings of the 1st Internet of Services Doctoral Symposium 2008 at International Conference on Interoperability of Enterprise Systems and Applications (I-ESA 2008)*. Edited by R. Ruggaber. Volume 374. CEUR Workshop Proceedings. 2008.

[Kne08b] M. Knechtel. "Access rights and collaborative ontology integration for reuse across security domains." In: *Proceedings of the ESWC 2008 Ph.D. Symposium*. Edited by P. Cudré-Mauroux. Volume 358. CEUR Workshop Proceedings. *Best Poster Award.* 2008, pages 36–40.

Chapter 3 provides results from representing and completing RBAC policies. In Section 3.1 we discuss how DLs can be applied to represent knowledge from an RBAC policy. We argue that this makes sense since common tasks related to RBAC policies can be reduced to standard DL reasoning tasks. This includes computing implicit knowledge from explicit knowledge about permissions and querying for an access control decision. Both tasks are implemented in the architecture component to compute the *RBAC matrix* from the *DL representation*. However, this does not create any new knowledge that was neither explicit nor implicit before. In Section 3.2 we discuss how to *complete* RBAC policies in the sense of adding new knowledge that has not yet been represented in a computer or followed from this representation but has been known to an expert, e.g. the security engineer. This is a knowledge acquisition task. In a systematic, non-redundant computer-supported dialog with this expert, the knowledge about an RBAC policy is completed. One part is obtaining a *user role hierarchy* in the form

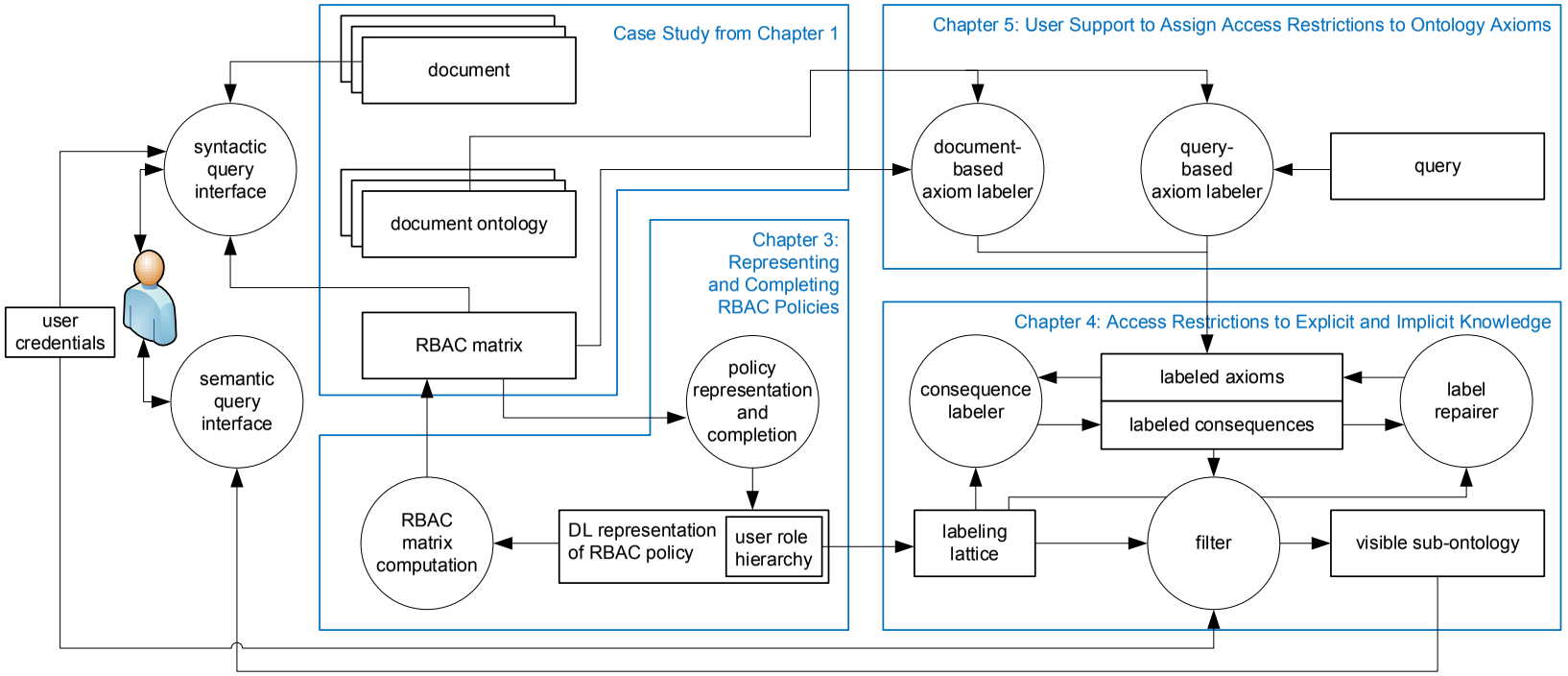

Figure 1.7: Architecture of an access restricted semantic document store

of a lattice, which can be used as the basis for the labeling lattice in the next chapter. In Section 3.3 we extend the completion of RBAC policies so that implications known a priori are exploited in order to pose fewer questions to the security engineer. Most of the results have been published in the following papers and a filed patent:

[DK09] F. Dau and M. Knechtel. "Access Policy Design Supported by FCA Methods." In: *Proceedings of the 17th International Conference on Conceptual Structures (ICCS 2009)*. Edited by F. Dau and S. Rudolph. Volume 5662. Lecture Notes in Computer Science. 2009, pages 141–154. (Cited on page 38).

[DK10] F. Dau and M. Knechtel. "Systems and Methods for Generating Constraints for use in Access Control." Patent (United States). Application Number 12/823,884. Date of application 25.6. 2010. (Cited on page 38).

[KH08] M. Knechtel and J. Hladik. "RBAC Authorization Decision with DL Reasoning." In: *Proceedings of the IADIS International Conference WWW/Internet (ICWI 2008)*. Edited by P. Isaías, M. B. Nunes, and D. Ifenthaler. 2008, pages 169–176. (Cited on pages 38, 39).

[KHD08] M. Knechtel, J. Hladik, and F. Dau. "Using OWL DL Reasoning to decide about authorization in RBAC." In: *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*. Edited by C. Dolbear, A. Ruttenberg, and U. Sattler. Volume 432. CEUR Workshop Proceedings. 2008. (Cited on pages 38, 39).

Chapter 4 provides results for access restrictions to explicit (and implicit) knowledge represented in an ontology (and following from that ontology). The basis is an ordered criterion for access restrictions which can be represented with a *labeling lattice*. One instance is a set of user roles and the user role hierarchy, which can express that some user role has at least all permissions of another user role. In Section 4.1 we introduce *labeled axioms* which are axioms that receive an element from the labeling lattice as a label. Users also receive a label from that lattice, e.g. representing their assigned user roles, and a user is allowed to see an axiom if her label dominates the axiom label. Based on this label comparison, a *filter* delivers the *visible sub-ontology* to a user. In Section 4.2 we show that the label for a consequence, called the boundary, can be computed from the labels of the axioms. In Section 4.3 we introduce how to *repair* the boundary. This becomes necessary, if the security administrator wants to change the computed boundary to another label. The only way to achieve this is by changing some axiom labels. We show how to compute a change set, which is a set of axioms that require a new label in order to yield the desired boundary. This problem is further generalized to repairing multiple boundaries simultaneously in Chapter 5. All our algorithms are black-box based, i.e. they can be used with any off-the-shelf reasoner without the need for modifications. This means, access restrictions to an ontology can be enforced without special reasoners. Most of the results have been published in:

[BKP09a] F. Baader, M. Knechtel, and R. Peñaloza. *Computing Boundaries for Reasoning in Sub-Ontologies*. LTCS-Report 09-02. Available at `http://lat.inf.tu-dresden.de/research/reports.html`. Germany: Chair of Automata Theory, Institute of Theoretical Computer Science, Technische Universität Dresden, 2009. (Cited on page 61).

[BKP09b] F. Baader, M. Knechtel, and R. Peñaloza. "A Generic Approach for Large-Scale Ontological Reasoning in the Presence of Access Restrictions to the Ontology's Axioms." In: *Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*. Edited by A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan. Volume 5823. Lecture Notes in Computer Science. 2009, pages 49–64. (Cited on pages 61, 85).

[KP10a] M. Knechtel and R. Peñaloza. "A Generic Approach for Correcting Access Restrictions to a Consequence." In: *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*. Edited by L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache. Volume 6088. Lecture Notes in Computer Science. 2010, pages 167–182. (Cited on pages 61, 85).

[KP10b] M. Knechtel and R. Peñaloza. "Correcting Access Restrictions to a Consequence." In: *Proceedings of the 23rd International Workshop on Description Logics (DL 2010)*. Edited by V. Haarslev, D. Toman, and G. Weddell. Volume 573. CEUR Workshop Proceedings. 2010, pages 220–231. (Cited on pages 61, 85).

Chapter 5 provides results which facilitate practical applications with the framework of Chapter 4, since the goal of this dissertation is not only to provide a theoretically solid, but also a practically usable framework. Security administrators might (yet) be unfamiliar with the task of assigning access restrictions to ontology axioms, so we introduce two techniques. In Section 5.1 we introduce *document-based* access restrictions. The idea is to reuse access restrictions which have been defined by the RBAC matrix for document types. We show how to apply them to the respective document ontology and all contained axioms. It might happen that the same axiom is contained in several document ontologies where the intuition would be that a user is allowed to read the axiom if she is allowed to read at least one of those documents. In Section 5.2 we show how to describe access restrictions intentionally by means of a query, called *query-based* access restriction. All results of that query, including implicit consequences and their intended label are collected in a goal set. It can be enforced by a changed axiom labeling specified by a change set. We show that this problem is in fact a generalization of repairing one consequence's boundary from Chapter 4 and that it can be solved with an extension. For goal sets containing a conflict, we provide two conflict resolution strategies. Alternatively to our label filtering framework, query-based access restrictions could be enforced by query rewriting. In a comparison, we show that label filtering is independent of a concrete ontology language and has higher knowledge availability in the sense that more answers are delivered to a user

that are uncritical since they do not uncover any secret. Most of the results have been published in:

[KS08] M. Knechtel and D. Schuster. "Semantische Integration und Wiederverwendung von Produktontologien für offene Marktplätze im Web." In: *Tagungsband des Gemeinschaften in Neuen Medien Workshops (GeNeMe 2008)*. Edited by K. Meißner and M. Engelien. In German. 2008, pages 177–188. (Cited on page 89).

[KS10] M. Knechtel and H. Stuckenschmidt. "Query-Based Access Control for Ontologies." In: *Proceedings of the 4th International Conference on Web Reasoning and Rule Systems (RR 2010)*. Edited by P. Hitzler and T. Lukasiewicz. Volume 6333. Lecture Notes in Computer Science. 2010, pages 73–87. (Cited on page 89).

Chapter 6 provides empirical results for the algorithms from Chapters 4 and 5 which show that our algorithms perform well in practical scenarios with large-scale ontologies. In Section 6.1 we describe our test data and environment covering the implementation language, the test PC system, the DL reasoners, the labeling lattices, the ontologies, and the assignment of labels in the labeled ontologies. Section 6.2 provides our results from experiments with the algorithms for computing a consequence's boundary, repairing this boundary and repairing the boundaries of several consequences simultaneously. Most of the results have been published in the papers mentioned above for Chapters 4 and 5.

Chapter 7 summarizes the technical and empirical results achieved during the PhD project and suggests directions for future work.

People can't share knowledge if they don't speak a common language.

Thomas H. Davenport

2 Preliminaries

This chapter introduces the preliminaries necessary to describe the technical contributions of this thesis. It is divided into four sections.

First, an introduction to KR with Description Logics deals with its history and applications with focus on the Semantic Web. It further deals with basic terminology on ontologies and reasoning, and services on explaining and debugging implicit knowledge.

Second, an introduction to lattices contains the definition as a partially ordered set and the equivalent definition as an algebraic structure. The mathematical theory of FCA is introduced since it can be used to describe and discuss objects of a domain with their attributes and to identify concepts and the hierarchical order amongst them in the form of a concept lattice.

In the last two sections, an introduction to access control, applicable for systems with explicit knowledge, and inference control, applicable for systems with implicit knowledge, is given. The access control matrix is introduced as basic model for access control systems and based on that, details for lattice-based access control (LBAC) and RBAC are introduced. For inference control, two basic mechanisms are introduced: one applicable while the system is running and one applicable before the system is running.

2.1 Knowledge Representation with Description Logics

KR is an area of artificial intelligence with the goal of representing knowledge in a manner, so that computers can process and use it. Reasoning is the task of inferring implicit knowledge from explicit knowledge. A range of formalisms with different expressivity and reasoning problems of different complexity are available. This includes propositional logic, first-order logic, Semantic Networks, conceptual graphs, Topic Maps, Semantic Web standards RDF, RDFS, OWL, etc. [HLP07].

DL systems [Baa+07b] are formal knowledge representation and reasoning systems which provide inference services that deduce implicit knowledge from the explicitly rep-

resented knowledge. For these inference services to be practically feasible the underlying inference problems must at least be decidable, which is one of the characteristics of DLs.

In this section we bring DLs into a historical context and introduce some applications. We define important notions for DL systems and introduce explanation and debugging services. These services help to understand consequences and to detect and remove modeling errors contained in the explicit knowledge causing non-intended consequences.

History and Application

DLs evolved from the early KR formalisms *Semantic Networks* and *Frames*, both already introducing the notion of classes of individuals and relations between the classes.

In Semantic Networks [Qui67], a class or an individual is realized by a vertex and a relation is realized by a labeled edge. The special *is-a* relation relates two classes or an individual and a class. Relations are inherited along is-a edges.

In Frame systems [Min81], classes are realized as Frames where each Frame has a name, a collection of more general Frames, and a collection of slots. The slots specify relations to other classes, similar to the edges in Semantic Networks.

The problem of both, Semantic Networks and Frames, is the lack of formally defined semantics. The meaning of a given knowledge representation is left to the intuition of the programmer who builds a reasoner. Consequently for the *is-a* relation there are at least 6 different meanings for an is-a relation between two classes, and at least 4 different meanings for an is-a relation between an individual and a class [Bra83]. Figure 2.1 provides an example Semantic Network. Since vertices may represent classes or individuals, it is not necessarily clear that Kermit was intended to be an individual frog while CommonFrog is a subclass of the class Frog. The has-color edge can be read with two different interpretations: (1) the color of frogs is green only or (2) frogs have (among others) the color green. In many systems, inheritance is only by default, i.e. as long as no conflict arises: for example the CommonFrog is brown, overriding a default color green of Frog. Due to the lack of formal semantics, different reasoning algorithms for the same formalism could yield different results on the same knowledge representation. This can be avoided by a declarative semantics, defined formally and independently of a specific reasoner. For Semantic Networks and Frames, the semantics introduced in [SGC79; Hay79] employed a relatively small fragment of first-order logic. Based on these KR formalisms, *logic-based concept languages* were developed which are now known as *Description Logics*.

Description Logics are today embodied in many knowledge-based systems and have been used to develop various real-life applications. The most popular application is the Semantic Web. The W3C developed and recommended OWL as the standard ontology language for the Semantic Web [MPSP09], and DLs provide its basis. We will go into more detail of the Semantic Web below. A variety of further applications, from natural language processing, configuration of technical systems, software information systems, heterogeneous databases integration, support for planning, etc., is described, e.g., in the Description Logic Handbook [Baa+07b]. The most notable application outside

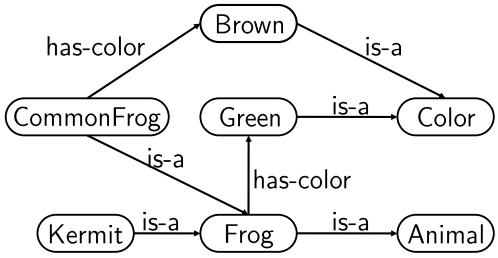

Figure 2.1: An example Semantic Network

information science is in bioinformatics where medical knowledge is codified with DLs. For example, the SNOMED CT[1] ontology contains almost 400,000 axioms and more than five million consequences follow from that axioms. The ontology is used in more than forty countries around the world.

Some state-of-the-art DL reasoners, available and continuously maintained until today, are RACER [HM01], KAON 2 [HMS04], FaCT++ [TH05], CEL [BLS06], Pellet [Sir+07], and HermiT [MSH07].[2]

Description Logics in the Semantic Web

In the remaining section, we go into more detail of the Semantic Web and its application of DLs. It has long been realized that the Web could benefit from having its content made available in a machine processable form, which enables computers to interpret data. While the Web language HTML is focused on presentation and text formatting rather than content, languages such as XML do add some support for capturing the meaning of Web content. The Semantic Web has been envisioned as an evolution from a linked document repository into a platform where "information is given well-defined meaning, better enabling computers and people to work in cooperation" [BLHL01] and, to limit the scope, which "will enable machines to COMPREHEND [*original emphasis*] semantic documents and data, not human speech and writings" [BLHL01]. This is to be achieved by augmenting the existing layout information with semantic annotations that add descriptive terms to Web content, with the meaning of such terms being defined in ontologies. The DARPA Agent Markup Language (DAML) and Ontology Inference Layer (OIL) ontology languages for the Semantic Web are syntactic variants of DLs [Hor02] and have been the starting point for the W3C Web Ontology Working Group. They finished their work in 2004 with the publication of the OWL standard [Bec+04]. In 2007 the W3C OWL Working Group began working on refinements and extensions to OWL, and finished in 2009 with the publication of the OWL 2 standard [MPSP09].

In order to enable reuse of existing content and to minimize manual effort of semantic annotations, methods for information extraction have been investigated. Intuitively, those methods try to make good guesses based on statistical data, document structure

[1] `http://www.ihtsdo.org/snomed-ct/`

[2] For a list of DL reasoners see, e.g., `http://www.cs.man.ac.uk/~sattler/reasoners.html`.

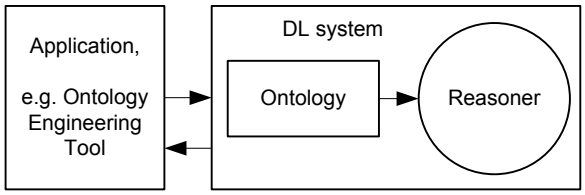

Figure 2.2: Basic architecture of a DL system interacting with an application

and language structure. For example, somebody not able to read Korean can still guess that on the Korean Wikipedia page about the city Dresden, the number followed by the unit "km^2" is the city's area, the number followed by "m" is the city's elevation above mean sea level, etc. The accuracy of those methods is intrinsically below 100%. Basic named entities can be extracted with close to 90% accuracy, relations can be extracted with an accuracy currently not much higher than 70% even in restricted domains [Sar08].

Description Logic Terminology

The purpose of this section is to introduce the syntax and semantics of DLs in compact manner. For more details we refer the reader to [Baa+07b]. We will first introduce the basic Description Logic $\mathcal{ALC}$ and then describe several of its extensions, specifically those used in Semantic Web standards.

The basic components of a DL-based knowledge representation system, for short DL system, are illustrated in Figure 2.2. DL systems consist of an ontology which represents explicit knowledge and a reasoner which makes implicit consequences of this knowledge explicit. The explicit and implicit knowledge is exploited by the application by interacting with the DL system.

A DL is a symbolic language defining concepts, individuals and relationships among them. Intuitively, concepts represent classes of individuals and roles represent relations between individuals. For a basic DL, called *attributive language with complement* $\mathcal{ALC}$, we define syntax and semantics in the following.

Definition 1 (Syntax of $\mathcal{ALC}$)**.** Let CN, IN, RN be disjoint sets of concept names, individual names and role names, respectively. Each individual name $a \in \mathsf{IN}$ is an *individual*. Each role name $P \in \mathsf{RN}$ is a *role description* (*role* for short). *Concept descriptions* (*concepts* for short) in $\mathcal{ALC}$ are inductively defined as:

- each concept name $A \in \mathsf{CN}$, bottom $\bot$ and top $\top$ are concepts
- if C, D are concepts, and P is a role, then $\neg C$ (negation), $C \sqcap D$ (intersection), $C \sqcup D$ (union), $\forall P.C$ (value restriction), $\exists P.C$ (existential quantification) are concepts

Example 2. With the given syntax for concept and role descriptions we can, e.g., describe digital cameras. Suppose that $DigitalCamera, SLR$ and $Zoom$ are concepts.

Then $DigitalCamera \sqcap SLR$ and $DigitalCamera \sqcap \neg SLR$ are $\mathcal{ALC}$-concepts, intuitively describing those digital cameras that are single-lens reflex cameras and those that are not single-lens reflex cameras. If we suppose in addition that $hasComponent$ is a role, we can form the concepts $DigitalCamera \sqcap \exists hasComponent.Zoom$ and $DigitalCamera \sqcap \forall hasComponent.\neg Zoom$, describing digital cameras that have a zoom, and those cameras that have no zoom.

Whereas syntax defines which collection of symbols are legal expressions, semantics defines the meaning of these expressions. The name "Description *Logics*" already indicates a formal logic-based semantics. Like any DLs, the semantics of $\mathcal{ALC}$ concept descriptions is defined through interpretations. Intuitively, the interpretation of a concept is the set of all individuals that belong to that concept and the interpretation of a role is a binary relation between individuals.

Definition 2 (Semantics of $\mathcal{ALC}$)**.** An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the *domain* of the interpretation) and an *interpretation function* $\cdot^{\mathcal{I}}$, which assigns to every individual name a an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, to every concept name A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every role name P a binary relation $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the following inductive definitions:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall P.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.\ (a,b) \in P^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists P.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a,b) \in P^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}.
\end{aligned}
$$

The concept and role descriptions are not enough to formulate statements about our world. For example, we might want to say "a single-lens reflex camera is a camera." The set of allowed axioms and their syntax and semantics is defined for each DL. For $\mathcal{ALC}$ the following axioms are allowed.

Definition 3 (Syntax of $\mathcal{ALC}$ axioms)**.** Let a, b be individual names, P be an $\mathcal{ALC}$ role description and C, D be $\mathcal{ALC}$ concept descriptions. *Terminological $\mathcal{ALC}$ axioms* have the form of a *general concept inclusion* $C \sqsubseteq D$ or an *equivalence* $C \equiv D$. *Assertional $\mathcal{ALC}$ axioms* have the form of a *concept assertions* $C(a)$ or a *role assertion* $P(a, b)$.

Again, the semantics of $\mathcal{ALC}$ axioms is defined through interpretations.

Definition 4 (Semantics of $\mathcal{ALC}$ axioms)**.** The interpretation $\mathcal{I}$ *satisfies* (*is a model of*) the general concept inclusion (GCI) $C \sqsubseteq D$ if it fulfills the semantics condition $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, the equivalence $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$, the concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and the role assertion $P(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$, respectively. We write $\mathcal{I} \models \alpha$ if an interpretation $\mathcal{I}$ satisfies an axiom α.

We call a set of axioms an *ontology*. For the Semantic Web community, [RS07] gives the following definition based on [Gru93]. The additions compared to [Gru93] are in bold text.

> An ontology is a **formal** explicit specification of a **shared** conceptualisation **of a domain of interest**.

In our context we stick to a much more precise definition in this thesis, which still allows the above generic definition. A set of axioms of allowed syntax forms an *ontology*, also called a *knowledge base*. An ontology has a model if all of its axioms can be satisfied at the same time. Formally, we give the following definition:

Definition 5 (Ontology)**.** A DL *ontology* O is a finite set of axioms, with disjoint subsets T, A of terminological axioms (often called TBox) and assertional axioms (often called ABox). Let $\mathcal{I}$ be an interpretation, then $\mathcal{I}$ *satisfies* (also *is a model of*) O, written as $\mathcal{I} \models O$, if for each axiom $\alpha \in O, \mathcal{I} \models \alpha$. Two axioms or two sets of axioms are equivalent if they have the same models.

The TBox expresses how concepts or roles are related to each other. The ABox introduces names for individuals and the restrictions that they have to satisfy. The TBox contains vocabulary which is used to describe a concrete world in the ABox. In a simplified view, an ABox can be seen as an instance of a relational database with only unary or binary relations. However, while a database instance represents exactly one interpretation, namely the one where classes and relations in the schema are interpreted by the objects and tuples in the instance, an ABox represents many different interpretations, namely all its models. The former is called "closed-world semantics", the latter "open-world semantics." The difference is that absence of information in a database instance is interpreted as negative information, while absence of information in an ABox only indicates a lack of knowledge. Moreover, the TBox imposes semantic relationships between the concepts and roles in the ABox that do not have counterparts in database semantics.

As stressed in the introduction, an important motivation for building a knowledge representation is to identify implicit consequences following from the explicit knowledge. Consequences following from a DL ontology have the same structure as axioms, but they are not necessarily contained in the ontology. Formally, a consequence is defined as follows.

Definition 6 (Consequence)**.** An axiom c *follows from*[3] an ontology O, written as $O \models c$ if for all models $\mathcal{I}$ of O if holds that $\mathcal{I} \models c$.

If a consequence follows from an ontology, we say that the ontology *entails* the consequence.

The Defintion 5 and Definition 6 together imply that every axiom contained in an ontology follows from this ontology. Conversely, $O \models c$ does obviously not necessarily mean $c \in O$. Note that this is a rather restricted definition of a consequence. Literature considers more types of consequences such as, e.g., the fact that a concept C is satisfiable ($\exists \mathcal{I}.\ C^{\mathcal{I}} \neq \emptyset$) or that an ontology O is consistent ($\exists \mathcal{I}.\ \mathcal{I} \models O$). We use this restricted definition only. To give an example, a subsumption $C \sqsubseteq D$ follows from O iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of O. For a second example, an instance relation $C(a)$ follows from O iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of O.

[3] Literature also often uses the terms *is consequence of*, *is entailed by* or *is implied by* an ontology.

OWL 2 functional syntax	DL syntax	Semantics	Symbol
owl:Thing	$\top$	$\Delta^{\mathcal{I}}$	$\mathcal{AL}$
owl:Nothing	$\bot$	$\emptyset$	$\mathcal{AL}$
ObjectIntersectionOf(C D)	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$\mathcal{AL}$
ObjectUnionOf(C D)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\mathcal{U}$
ObjectComplementOf(C)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\mathcal{C}$
ObjectAllValuesFrom(P C)	$\forall P.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.\ (a,b) \in P^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	$\mathcal{AL}$
ObjectSomeValuesFrom(P C)	$\exists P.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a,b) \in P^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	$\mathcal{E}$
ObjectMinCardinality(n P C)	$\leqslant n\ P.C$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in C^{\mathcal{I}} \mid (a,b) \in P^{\mathcal{I}}\}\rvert \leqslant n\}$	$\mathcal{Q}$
ObjectMaxCardinality(n P C)	$\geqslant n\ P.C$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in C^{\mathcal{I}} \mid (a,b) \in P^{\mathcal{I}}\}\rvert \geqslant n\}$	$\mathcal{Q}$
ObjectExactCardinality(n P C)	$= n\ P.C$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in C^{\mathcal{I}} \mid (a,b) \in P^{\mathcal{I}}\}\rvert = n\}$	$\mathcal{Q}$
ObjectOneOf(a)	$\{a\}$	$\{a^{\mathcal{I}}\}$	$\mathcal{O}$

Table 2.1: Some OWL 2 concept constructors [Baa+07b; MPSP09]

The relation $\models$ between ontologies and consequences is called *consequence relation*. This relation is monotonic for DLs, i.e. for every ontology O, we have that if $O' \subseteq O$ and $O' \models c$, then $O \models c$.

Example 3. If a knowledge engineer feels she is not able to define the concept "digital SLR" in all detail, she can require that a digital SLR is a SLR with the GCI $DSLR \sqsubseteq SLR$. If she is able to define it completely, she could state the equivalence $DSLR \equiv SLR \sqcap \exists projectsOn.Sensor$. Given two individual names *Canon550D* and *CMOS550D*, we can say that Canon 550D is a digital SLR by the concept assertion $DSLR(Canon550D)$ and projects on a CMOS 550D sensor by the role assertion $projectsOn(Canon550D, CMOS550D)$. Let O be an ontology consisting of two axioms $O = \{DSLR(Canon550D), DSLR \sqsubseteq SLR\}$ from which, e.g., the consequence $c_1 : SLR(Canon550D)$ and the consequence $c_2 : DSLR(Canon550D)$ follow. While c_2 is contained explicitly in O (and trivially also follows from O), c_1 only follows from O.

There are several possibilities for extending $\mathcal{ALC}$ in order to obtain a more expressive Description Logic. Syntax and semantics of some concept constructors and some axiom constructors, including those for $\mathcal{ALC}$, are given in Table 2.1 and in Table 2.2, where C, D are concept descriptions, P, Q, R are role descriptions, a, b, c are individual names and n is a natural number. The tables provide the OWL 2 functional syntax for readers who are familiar with OWL 2 already. In this thesis, the DL syntax is used.

The letter $\mathcal{S}$ stands for the DL $\mathcal{ALC}$ extended by transitive roles. The Description Logic $\mathcal{SROIQ}(\mathbf{D})$ is the formal basis of the W3C OWL 2 standard [HKS06; MPSP09]. Based on $\mathcal{S}$, it additionally supports complex role inclusions ($\mathcal{R}$), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$) and qualified number restrictions ($\mathcal{Q}$). Additionally, the logic can be parameterized by one or more concrete domains ($\mathbf{D}$) which correspond to data types in OWL and permit reference to concrete data objects such as strings and integers. For example, "persons over 18" can be described with the concept description $Person \sqcap hasAge.\geqslant_{18}$, where the unary predicate $\geqslant_{18}$ is defined on the set of positive integers.

OWL 2 functional syntax	DL syntax	Semantics	Symbol
SubClassOf(C D)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	
EquivalentClasses(C D)	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$	
ClassAssertion(C a)	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$	
ObjectPropertyAssertion(P a b)	$P(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$	
InverseObjectProperties(P R)	$P \equiv R^-$	$P^{\mathcal{I}} = \{(a, b) \mid (b, a) \in R^{\mathcal{I}}\}$	$\mathcal{I}$
SubObjectPropertyOf(P R)	$P \sqsubseteq R$	$P^{\mathcal{I}} \subseteq R^{\mathcal{I}}$	$\mathcal{H}$
SubObjectPropertyOf(ObjectPropertyChain(P Q) R)	$P \circ Q \sqsubseteq R$	$\{(a, c) \mid \exists b.\ (a, b) \in P^{\mathcal{I}} \wedge (b, c) \in Q^{\mathcal{I}}\} \subseteq R^{\mathcal{I}}$	$\mathcal{R}$

Table 2.2: Some OWL 2 axiom constructors [Baa+07b; MPSP09]

However, there is no reason for a higher expressivity than required by the application, and often smaller means better reasoning performance. One part of the OWL 2 standard is the OWL 2 EL profile [MPSP09] based on the lightweight Description Logic $\mathcal{EL}^{++}$ [BBL05]. The concept constructors supported by the Description Logic $\mathcal{EL}$ are intersection, existential quantification (but not value restriction) and top concept $\top$. Additionally, the Description Logic $\mathcal{EL}^{++}$ supports the bottom concept $\bot$, nominals, concrete domains and role inclusion axioms.

The OWL standard calls a concept a *class*, a role a *property* and an individual an *object*. In the following, we will often use the OWL terminology in order to clearly distinguish DL roles from user roles.

A detailed introduction, also to further members of the Description Logics family and extensions can, e.g., be found in the Description Logic Handbook [Baa+07b].

Explanation and Debugging

In this thesis, explanation and debugging techniques turn out to be the basis to compute and to repair a consequence's label. Similar to writing large software systems, building large-scale ontologies is error-prone. An ontology might imply unexpected or even undesired consequences. A real-world example of an unintended consequence is the subsumption relationship "amputation of finger is an amputation of arm" which follows from the SNOMED CT ontology [Sun08]. However, finding a reason, i.e. a set of responsible axioms, by just looking at the 400,000 axioms manually is not realistic. Humans are usually not good in seeing implications from large sets of axioms.

An *explanation* for a consequence following from an ontology is a minimal set of axioms from the ontology from which the consequence still follows. Formally, it is defined as follows.

Definition 7 (MinA)**.** Let O be an ontology and c a consequence such that $O \models c$. A subset $S \subseteq O$ is a *minimal axiom set (MinA)* (also *explanation*) for $O \models c$ if $S \models c$ and $S' \not\models c$ for every $S' \subset S$.

The dual notion of a MinA is that of a diagnosis, which is a minimal set of axioms which need to be removed so that a consequence does not follow anymore. Formally, it is defined as follows.

Definition 8 (Diagnosis)**.** Let O be an ontology and c a consequence such that $O \models c$. A subset $S \subseteq O$ is a *diagnosis* for $O \models c$ if $O \setminus S \not\models c$ and $O \setminus S' \models c$ for every $S' \subset S$.

Each MinA (diagnosis) is minimal with respect to set inclusion, i.e. no axiom can be dropped without losing (regaining) the consequence. However, there might be several MinAs (diagnoses) and so a given MinA (diagnosis) is not necessarily minimal with respect to set cardinality.

Example 4. For two of the consequences following from the ontology in Example 1, the MinAs and diagnoses are the following. The first considered consequence $c_1 : \mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$ follows from each of the MinAs $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}, \{a_1, a_3, a_5\}$. Similarly, the second considered consequence $c_2 : \mathit{LowProfitService}(\mathit{ecoCalculatorV1})$ follows from each of the MinAs $\{a_1, a_2\}, \{a_1, a_3\}$. The consequence c_1 does not follow anymore when removing any of the diagnoses $\{a_1\}, \{a_2, a_3\}, \{a_4, a_5\}$ from the ontology. Similarly, the consequence c_2 does not follow anymore when removing any of the diagnoses $\{a_1\}, \{a_2, a_3\}$.

Technically, there are two approaches for computing MinAs. The *glass-box approach* takes a specific reasoner (or reasoning technique) for an ontology language (e.g., a tableau-based reasoner for OWL 2 [Sir+07]) and modifies it such that it can compute MinAs [SC03; Mey+06; Kal+05; Kal+07; BP08; BP10b]. Glass-box approaches might have high optimization potential, but the problem is that they have to be developed for every ontology language and reasoning technique anew and that optimizations of the original reasoning technique do not always apply to the modified reasoners. In contrast, the *black-box approach* can re-use existing highly-optimized reasoners without modifications, and it can be applied to arbitrary ontology languages: one just needs to plug in a reasoner for this language [Kal+05; Kal+07; BS08; BPS07; Sun08].

The task of computing all MinAs, also called *axiom pinpointing*, has been widely studied in recent years, and there exist implementations for a black-box approach [Kal+05; Kal+07; Sun08] based on Reiter's Hitting Set Tree (HST) algorithm [Rei87]. When we extend the HST-based algorithm for axiom pinpointing in Section 4.2.4, we will introduce its details more comprehensively.

2.2 Lattices and Formal Concept Analysis

The theory of ordered sets and lattices provides a tool to discuss and analyze hierarchies which often appear in our real world. In this thesis, lattices are used to capture a criterion with its values and the order of the values. Elements of this lattice, called labeling lattice, define visible sub-ontologies. One criterion is the required user role so that an axiom is visible. In this case the set is a set of user roles and the order is the user role hierarchy. This section will introduce order-theoretic preliminaries and define a lattice as partial order and as algebraic structure. Since both definitions are equivalent, one may freely refer to aspects of either definition fitting best to one's purpose.

We will then introduce a formalism to describe objects of a domain with their attributes. As it will turn out later, the access control matrix of an information system can be interpreted as such a context. With methods from FCA, a context can

be analyzed by identifying concepts and their hierarchical order, and by identifying implications between attributes. Those implications can be explored and either confirmed or refuted during an interactive dialog with a domain expert. In this thesis, we use methods from FCA in order to explore constraints from an RBAC matrix.

Lattices

We first define some order-theoretic preliminaries needed for the definition of a lattice.

Definition 9 (Partial order)**.** Let P be a set. A binary relation $\leqslant$ on P is a *partial order*, if for all elements $x, y, z \in P$ the following conditions hold:

1. $x \leqslant x$ (reflexivity),
2. $x \leqslant y$ and $y \leqslant x$ implies $x = y$ (antisymmetry),
3. $x \leqslant y$ and $y \leqslant z$ implies $x \leqslant z$ (transitivity).

A set P together with a partial order relation $\leqslant$ is a *partially ordered set*. For distinct elements a, b of a partially ordered set P, if $a \leqslant b$ or $b \leqslant a$ then a and b are *comparable*, otherwise they are *incomparable*. If any two elements are comparable, the order relation is called a *total order*, also *linear order*.

Definition 10 (Join, meet)**.** Let P be an ordered set and let $S \subseteq P$. An element $x \in P$ is an *upper bound* of S iff for all $s \in S : s \leqslant x$ holds. Dually, an element $x \in P$ is a *lower bound* of S iff for all $s \in S : x \leqslant s$ holds. There might be a least (greatest) element in the set of all upper bounds (lower bounds). An element $x \in P$ is the *least upper bound* (also known as *join* or *supremum*) of S if

1. x is an upper bound of S, and
2. $x \leqslant y$ for all upper bounds y of S.

The *greatest lower bound* (also known as *meet* or *infimum*) is defined dually. Both, join and meet are unique when they exist. When it exists, we write $\bigoplus S$ for the supremum of S, or $a_1 \oplus \ldots \oplus a_n$ with $S = \{a_1, \ldots, a_n\}$. Dually, we write $\bigotimes S$ for the infimum of S, or $a_1 \otimes \ldots \otimes a_n$.

Note that in literature, join is often denoted by $\wedge$ and meet is often denoted by $\vee$ [DP02]. We use the notation above, in order not to mix them up with Boolean operators.

With the given preliminary definitions, two equivalent definitions are available for a lattice. We will first define a lattice as a partially ordered set, and then as an algebraic structure.

Definition 11 (Lattice as partially ordered set)**.** A lattice $(L, \leqslant)$ is a set L together with a partial order $\leqslant$ on the elements of L such that a finite subset $S \subseteq L$ always has a join $\bigoplus S$ and a meet $\bigotimes S$.

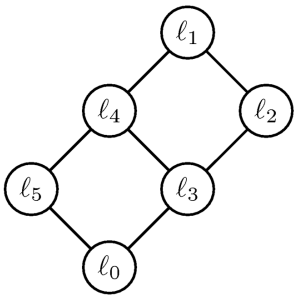

Figure 2.3: A lattice

If $\bigoplus S$ and $\bigotimes S$ exist for every $S \subseteq L$ then $(L, \leqslant)$ is a *complete lattice*.

Apart from this order-theoretic definition there is a definition as algebraic structure, since the two definitions of a lattice are equivalent, one may freely refer to aspects of either definition fitting best to one's purpose. We are emphasizing this here since literature on access control with lattices often uses only one of the two definitions.

Definition 12 (Lattice as algebraic structure)**.** An algebraic structure $(L, \oplus, \otimes)$ consisting of a set L and two binary operations $\oplus, \otimes$ on L is a lattice if the following axioms hold for all $a, b, c \in L$.

Commutative laws:	$a \oplus b = b \oplus a$,
	$a \otimes b = b \otimes a$
Associative laws:	$a \oplus (b \oplus c) = (a \oplus b) \oplus c$,
	$a \otimes (b \otimes c) = (a \otimes b) \otimes c$
Absorption laws:	$a \oplus (a \otimes b) = a$
	$a \otimes (a \oplus b) = a$

The two absorption laws together imply the idempotent laws $a \oplus a = a$ and $a \otimes a = a$, because $a \oplus a = a \oplus (a \otimes (a \oplus a)) = a$ and $a \otimes a = a \otimes (a \oplus (a \otimes a)) = a$.

Both definitions can be reduced to each other since the ordering $\leqslant$ can be recovered from the algebraic structure $(L, \oplus, \otimes)$ because $a \leqslant b$ holds iff $a = a \otimes b$ iff $b = a \oplus b$ for all elements $a, b \in L$.

A *bounded lattice* has a greatest (or maximum) and least (or minimum) element, denoted $\top$ and $\bot$ by convention, also called top and bottom. A bounded lattice is an algebraic structure of the form $(L, \oplus, \otimes, \bot, \top)$ such that $(L, \oplus, \otimes)$ is a lattice, $\bot$ is the identity element for the join operation $\oplus$, and $\top$ is the identity element for the meet operation $\otimes$. A finite lattice is automatically bounded with $\top = \bigoplus L$ and $\bot = \bigotimes L$. A complete lattice is automatically bounded with $\top = \bigotimes \emptyset$ and $\bot = \bigoplus \emptyset$.

Let $a, b, c \in L$, a is called *join prime* iff $a \leqslant b \oplus c \implies a \leqslant b$ or $a \leqslant c$.

Example 5. Figure 2.3 depicts a finite lattice $(L, \leqslant)$ with 6 elements. Every finite lattice is bounded, in this case by the greatest element ℓ_1 and least element ℓ_0. The order $\leqslant$ is represented by edges from bottom to top, for instance $\ell_5 \leqslant \ell_4$ and $\ell_5 \leqslant \ell_1$. Some joins are $\ell_2 \oplus \ell_3 = \ell_2, \ell_2 \oplus \ell_4 = \ell_1$, some meets are $\ell_2 \otimes \ell_3 = \ell_3, \ell_2 \otimes \ell_4 = \ell_3$. Join prime elements in L are $\ell_0, \ell_2, \ell_3, \ell_5$. Since $\ell_4 \leqslant \ell_3 \oplus \ell_5$ but neither $\ell_4 \leqslant \ell_5$ nor $\ell_4 \leqslant \ell_3$, ℓ_4 is not join prime. From a similar argument it follows that ℓ_1 is not join prime.

	optical zoom	*digital zoom*	*underwater*
cam_0			
cam_1	×		
cam_2		×	
cam_3	×	×	
cam_4		×	×
cam_5	×	×	×

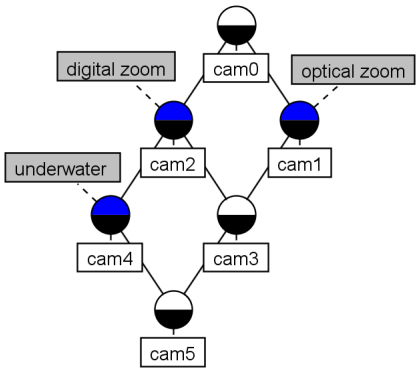

Figure 2.4: A context for 6 digital cameras and its concept lattice

A more detailed introduction to lattices and orders can, e.g., be found in [DP02].

Formal Concept Analysis

Hierarchies often appear in our world, for example in a company's organizational chart, in a family tree, in classes and subclasses of products in a catalog, etc. The theory of ordered sets and lattices provides a tool to discuss and analyze them. FCA is a mathematical theory which derives formal concepts from a given cross-table (i.e., a formal context) and investigates the hierarchical order amongst them.

Definition 13 (Formal context)**.** A *formal context* $\mathbb{K}$ is a triple (G, M, I) with G[4] the set of objects, M[5] the set of attributes and $I \subseteq G \times M$. Tuples (g, m) of the relation I are read as "g has attribute m."

We say *context* when we mean a formal context in the following. A finite context can be specified by a *cross-table* with one column per attribute and one row per object. A cross at row g and column m means $(g, m) \in I$. A missing cross means $(g, m) \notin I$.

Example 6. The left part of Figure 2.4 provides a cross-table that specifies a context (G, M, I) with 6 digital cameras and 3 attributes. For example, camera cam_2 has only the attribute *digital zoom*.

In traditional philosophy, a concept is understood as an entity consisting of two parts: the extension and the intension. The extension consists of all objects belonging to the concept and the intension is the collection of all attributes shared by the objects. Inspired by that, but with a slightly different naming in order to distinguish philosophy from the formal mathematical theory, a *formal concept* is determined by its *extent* and its *intent*. Formally, it is defined as follows.

Definition 14 (Formal concept)**.** Let $\mathbb{K} = (G, M, I)$ be a context. A *formal concept* of a context consists of an ordered pair (A, B), where A is a subset of G and called *extent*

[4]The letter was chosen from German: Gegenstände.
[5]The letter was chosen from German: Merkmale.

and B is subset of M and called *intent*. Objects in A are precisely those who share attributes in B. The set of all formal concepts of the context (G, M, I) is $\mathfrak{B}(G, M, I)$.[6]

Example 7. Let (G, M, I) be the context in Figure 2.4. A simple procedure to find a formal concept is the following: take an object (say cam_3) and let B be the set of its attributes ($B = \{$*optical zoom, digital zoom*$\}$). Let A be the set of objects which have all attributes in B ($A = \{cam_3, cam_5\}$). The obtained pair (A, B) is a formal concept $((\{cam_3, cam_5\}, \{$*optical zoom, digital zoom*$\}))$.

Definition 15 (Hierarchical order)**.** Let (G, M, I) be a context and $(A_1, B_1), (A_2, B_2)$ formal concepts of that context. The formal concept (A_1, B_1) is "less general" than the formal concept (A_2, B_2), written as $(A_1, B_1) \leqslant (A_2, B_2)$, iff the extent A_1 is contained in the extent A_2. Equivalently, $(A_1, B_1) \leqslant (A_2, B_2)$ iff the intent B_1 contains the intent B_2. That is,

$$(A_1, B_1) \leqslant (A_2, B_2) \Longleftrightarrow A_1 \subseteq A_2 \Longleftrightarrow B_1 \supseteq B_2.$$

The relation $\leqslant$ is the *hierarchical order* of formal concepts.

The order $\leqslant$ on the set of formal concepts defines a complete lattice, called the *concept lattice* $(\mathfrak{B}(G, M, I), \leqslant)$.

Example 8. Let (G, M, I) be the context in Figure 2.4. Let ℓ_3 be the formal concept $\ell_3 = (\{cam_3, cam_5\}, \{$*digital zoom,optical zoom*$\})$ and let ℓ_0 be the formal concept $\ell_0 = (\{cam_5\}, \{$*optical zoom,digital zoom,underwater*$\})$. The hierarchical order contains $\ell_0 \leqslant \ell_3$. The right part of Figure 2.4 depicts[7] the concept lattice for the context, consisting of all formal concepts and their hierarchical order. Gray boxes label the attributes, white boxes label the objects. Each node represents a formal concept consisting of all objects labeled below the node and all attributes labeled above the node. For example, the rightmost node represents the formal concept $\ell_2 = (\{cam_1, cam_3, cam_5\}, \{$*optical zoom*$\})$ and the lowest node represents the formal concept $\ell_0 = (\{cam_5\}, \{$*optical zoom,digital zoom,underwater*$\})$.

Given a context, another common method to analyze it is to find implications between attributes. They are constraints which hold in the context and are of the form "every object that has attributes $l_1, \ldots, l_i$ also has attributes $r_1, \ldots, r_j$."

Definition 16 (Implications between attributes)**.** Let $\mathbb{K} = (G, M, I)$ be a context. An *implication between the attributes* in M is a pair of sets $L, R \subseteq M$, usually written as $L \to R$. An implication $L \to R$ holds in $\mathbb{K}$ if every object in $\mathbb{K}$ that has all attributes in L also has all attributes in R. The set of all implications in $\mathbb{K}$ is $Imp(\mathbb{K})$.

Example 9. Based on an example in [Bur96], the context (G, M, I) in Figure 2.5 describes the natural numbers 1 to 9 with the following attributes: *even* (divisible by 2 without rest), *odd* (not divisible by 2 without rest), *prime* (only divisible by 1 and

[6] The letter was chosen from German: Begriffe.

[7] We used the Concept Explorer tool available at `http://conexp.sourceforge.net/` to generate and depict the concept lattice.

by itself, excluding 1), *square* ($= a^2$ for a natural number a) and *cubic* ($= a^3$ for a natural number a). One of the formal concepts is $(\{2, 3, 5, 7\}, \{prime\})$.

The minimal set of implications between the attributes of the given context,[8] but not necessarily all natural numbers (!) are

- $\{odd,\ cubic\} \to \{square\}$
- $\{square,\ cubic\} \to \{odd\}$
- $\{prime,\ cubic\} \to \{even,\ odd,\ square\}$
- $\{prime,\ square\} \to \{even,\ odd,\ cubic\}$
- $\{even,\ odd\} \to \{prime,\ square,\ cubic\}$.

So far we considered a context to contain all objects and attributes of interest. For example, we considered natural numbers 1 to 9, but not all natural numbers. However, also the concept lattice of a very large or even infinite context with a fixed set of attributes is determined by the implications between the attributes. For example, the concept lattice for a context with the objects being the infinite set of natural numbers can be determined. Ganter's interactive *attribute exploration* algorithm [Gan84] is a useful method to discuss implications between attributes with a domain expert in a computer supported dialog. In each step, the expert is asked whether a given implication holds. She confirms or provides a counterexample if it does not hold. The counterexample is added as a new object to the context. The context is changing during the dialog, but still the algorithm ensures that the expert does not have to answer redundant questions.

Example 10. The reader might have noticed already that two of the implications from Example 9 are true for the objects of the context (G, M, I) in Figure 2.5, but not for the set of all natural numbers. With regard to the chosen set of attributes, the numbers from 1 to 9 are not typical of all natural numbers. The implication $\{odd,\ cubic\} \to \{square\}$ does not hold, and one counterexample is 27 being *cubic* and *odd*, but not *square*. The implication $\{square,\ cubic\} \to \{odd\}$ does not hold and one counterexample is 64 being *square* and *cubic*, but not *odd*. The context extended with those two counterexamples is given in the left part of Figure 2.6. Only three of the five implications remain for the new context. They say that numbers being at the same time *prime* and *cubic* (respectively *prime* and *square*) are automatically *even* and *odd*. Humans know that a number cannot be *odd* and *even* by their definition, so there cannot be a number being *prime* and *cubic* or *prime* and *square*. The third implication says that an *even* and *odd* number is *prime*, *square* and *cubic*. Again, humans know that such a number does not exist by definition of *even* and *odd*. This is all consistent with the definition of the attributes, without the computer knowing their definition. The set of objects $G = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 27, 64\}$ is a typical set of objects with regard to the set of attributes M for natural numbers. The concept lattice of the resulting context is provided in the right part of Figure 2.6.

	even	*odd*	*prime*	*square*	*cubic*
1		×		×	×
2	×		×		
3		×	×		
4	×			×	
5		×	×		
6	×				
7		×	×		
8	×				×
9		×		×	

Figure 2.5: A context for some natural numbers

	even	*odd*	*prime*	*square*	*cubic*
1		×		×	×
2	×		×		
3		×	×		
4	×			×	
5		×	×		
6	×				
7		×	×		
8	×				×
9		×		×	
27		×			×
64	×			×	×

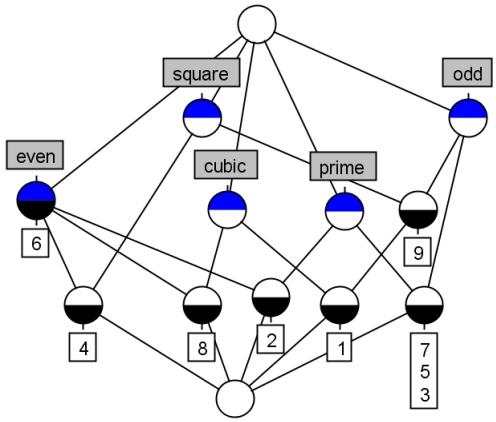

Figure 2.6: A context for natural numbers and its concept lattice

A more detailed introduction to FCA can, e.g., be found in [GW99].

Similarly to our examples where we discussed digital cameras (respectively natural numbers) and their attributes, we will discuss users and their allowed permissions.

2.3 Access Control

Organizations have a need to monitor their business activities and control access to their information systems. Access control is an essential operation in standard information systems to prevent unintended use of information in a system. It includes four tasks: identification, authentication, authorization, and accountability audit. Identification assigns an information system subject to a real user, for example by requesting a login name. Authentication makes sure that this assignment is correct, for example by requesting a password. Authorization determines what a subject can do in a system. Accountability audit identifies what a subject did. We focus on the *authorization* task only, when we refer to access control. In this thesis, we investigate access control methods in two aspects: (1) we apply it in order to protect explicit knowledge and (2) we investigate methods to represent and complete RBAC policies.

The expressiveness for declaring access rights can vary substantially, from an explicit representation and simple lookup, to a sophisticated knowledge representation from which access decisions logically follow. An access control matrix, first introduced by Lampson [Lam71], is a basic abstract formal computer security model. For any type of access control system it can model the static access permissions, ignoring further definitions of a policy like rules and dynamic behavior in time.

Definition 17 (Access control matrix)**.** Let S be a set of *subjects*, O a set of *objects*, A a set of *actions* and let $|\cdot|$ denote set cardinality. A matrix M with $|O|$ columns

[8] This is the Duquenne-Guigues-base of the implications valid in the context. Details can be found, e.g., in [GW99].

and $|S|$ rows where every element $M[s, o]$ for $s \in S, o \in O$ is a set of actions, is called *access control matrix*.

We say that a subject $s \in S$ is *allowed to* perform action $a \in A$ on object $o \in O$ iff $a \in M[s, o]$. An example for an access control matrix has been given in the case study in Section 1.2 already.

In practical systems, an access control matrix is often a sparse matrix, i.e. most matrix elements are empty. For this reason, more efficient data structures which furthermore facilitate building distributed systems are used in practical systems. For a given object, i.e. one matrix column, a list containing the permissions of all subjects on that object is called *access control list*. For a given subject, i.e. one matrix row, a representation of the set of permissions on all objects is called *capability*. This is not necessarily a list, but could be a more abstract representation. Examples, e.g. including a key from a public key infrastructure, are discussed in [And08].

The access control matrix is a basic model for different access control techniques. They can be categorized in

- discretionary access control (DAC),
- lattice-based access control (LBAC), also called mandatory access control (MAC),
- and role-based access control (RBAC).

In DAC systems, the permissions to access an object are defined and modified by its owner. In LBAC, the system determines the access to objects utilizing a lattice for assigning permissions to subjects. It thus removes the ability of the users to control access to their resources. RBAC systems can simulate DAC and LBAC, and they remove the explicit use of subjects and replace them with roles, which form a logical group of a number of subjects. In fact, permissions are assigned to roles and the subjects are assigned members of a number of roles. Thus changes of single subjects do not necessarily have consequences in the actual access control policies. The following sections go into more details of LBAC and RBAC.

Lattice-Based Access Control

Definitions in this section are introduced only as a preliminary step for RBAC, to facilitate understanding. Apart from that, they are not of elementary importance for this thesis.

MAC, called LBAC here following the terminology in [Den76; San93], is a technique determined by the information system. This is contrary to DAC which is determined by the owner of an object, such that each object of the access matrix has an assigned owner and she can change the respective column of the access matrix in order to change the object's permissions. LBAC was originally developed for military purposes, but is in general applicable to any situation where the flow of information has to be controlled [Den76; San93]. Each object receives a *security classification* and each subject receives a *security clearance*. A flow of information requires a flow of one security class to another. An *information flow policy* defines between which of the security classes an information flow is allowed, defined formally by Denning [Den76] as follows.

Definition 18 (Information flow policy)**.** An *information flow policy* is a triple $(SC, \rightarrow, \oplus)$ where SC is a set of *security classes*, $\rightarrow \subseteq SC \times SC$ is a binary *can-flow relation* on SC, and $\oplus : SC \times SC \rightarrow SC$ is a binary join operator on SC.

An information flow $A \rightarrow B$ means information *can flow* from A to B. As pointed out by Denning in [Den76], an information flow policy can form a finite lattice:

Definition 19 (Information flow lattice)**.** An information flow policy $(SC, \rightarrow, \oplus)$ forms a finite lattice $(SC, \rightarrow)$ iff

- $(SC, \rightarrow)$ is a partially ordered set
- SC is finite
- SC has a lower bound L such that $L \rightarrow A$ for all $A \in SC$
- $\oplus$ is a least upper bound operator on SC.

Based on an information flow lattice, specific LBAC models are available for different requirements. Some examples are discussed in [San93], including the Chinese Wall lattice, the Bell-LaPadula model and the Biba model. The Chinese Wall lattice prevents information flow resulting in conflicts of interest. For example, in a consulting company information from one customer must not be used for a competing customer. The Bell-LaPadula model ensures confidentiality in information flows. The Biba model ensures integrity for information flows and is, e.g., used in the Microsoft Windows Vista operating system in order to ensure that the Web browser and all downloaded software runs with low integrity level. System files have a high integrity level and cannot be changed by an installer of downloaded software without explicit user confirmation.

The intuition of Biba is to allow information flow only from high integrity to low integrity. Usually, high integrity is placed toward the top of the lattice and low integrity is placed toward the bottom, so that the permitted flow is from top to bottom, directly opposite to Denning's information flow lattice. However, the relation can be inverted to map one lattice to the other. Formally, the Biba model is defined [San93] as follows.

Definition 20 (Biba access control model)**.** Let $(L, \leqslant)$ be a lattice of integrity labels, S the set of subjects, O the set of objects, $A = \{r, w\}$ a fixed set of actions to read and write, M an access control matrix, $\mathsf{lab} : S \cup O \rightarrow L$ a static assignment of an integrity label to each subject and object. Given an object $o \in O$, a subject $s \in S$ is allowed to

- read iff $r \in M[s, o]$ and $\mathsf{lab}(s) \leqslant \mathsf{lab}(o)$ (simple integrity),
- write iff $w \in M[s, o]$ and $\mathsf{lab}(s) \geqslant \mathsf{lab}(o)$ (integrity confinement).

Role-Based Access Control

RBAC [SFK00] is a standardized model to indirectly assign permissions to users by user roles. The permissions of users are the union of all permissions of their assigned roles. In enterprise context the user roles often represent job descriptions of employees.

RBAC can be considered a superset of LBAC and of DAC, since RBAC introduces more flexibility while it can still simulate LBAC and DAC as shown in [OSM00; And08].

Sandhu et al. [San93] define four reference models for role-based access control. Among those, RBAC_0 defines a basic RBAC system and RBAC_1 augments RBAC_0 with role hierarchies. Since role hierarchies are almost inevitably included whenever roles are discussed in the literature and are also commonly implemented in systems that support roles [San93], we always mean RBAC_1 when referring to RBAC. The definition below is a simplification, leaving out the notion of sessions.

Definition 21 (RBAC)**.** The *RBAC model* has the following components:

- $U, \mathcal{R}, P$ the set of *users*, *roles*, *permissions* respectively
- $PA \subseteq \mathcal{R} \times P$ a many-to-many assignment relation of roles to permissions
- $UA \subseteq U \times \mathcal{R}$ a many-to-many assignment relation of users to roles
- $RH \subseteq \mathcal{R} \times \mathcal{R}$ a partial order on $\mathcal{R}$ called *user role hierarchy* also written as $\leqslant_{\mathcal{R}}$
- implicit permissions: $r_1 \leqslant_{\mathcal{R}} r_2$ (role r_1 inherits permissions from role r_2) and $(r_2, p) \in PA \implies (r_1, p) \in PA$

A concrete instance of the RBAC model, with concrete users, roles, permissions, etc. is called an *RBAC policy.*

The original definition has role hierarchy $\geqslant_{\mathcal{R}}$ so the more powerful roles are the greater elements instead of the lower elements in our definition, but that is just a convention and our variant fits better to our framework. RBAC has been proposed [SFK00] and accepted as NIST standard[9], and is today widely used in practical systems.

The user role hierarchy of RBAC adds further implicit permissions to the initial set of explicit permissions. There is a relation between the access control matrix and the RBAC model, discussed in detail in [SHV01]. For this thesis, we introduce the *RBAC matrix* as a static representation of an RBAC policy, which is useful for example to efficiently access explicit and implicit permissions. Intuitively, in the matrix each user role is a subject, permissions from RBAC are defined as a set of tuples of action and object, and actions allowed for roles on objects are the same in the RBAC policy and the RBAC matrix. Formally, it is defined as follows.

Definition 22 (RBAC matrix)**.** Let M be an access control matrix with S, O, A the set of subjects, objects, actions respectively. Let furthermore PA be the permission assignment relation and $\mathcal{R}, P$ be the set of roles and permissions respectively of an RBAC policy. The matrix M is an *RBAC matrix* for the RBAC policy if $S = \mathcal{R}, P \subseteq A \times O$ and $(a, o) \in P, (r, (a, o)) \in PA \iff a \in M[r, o]$ holds.

An RBAC matrix only exists for RBAC policies of certain structure. The definition requires the set of permissions P of the RBAC policy to be a set of ordered pairs of

[9] The current standard can be accessed at the NIST Web page `http://csrc.nist.gov/rbac/`

action and object. RBAC policies where P is of different structure have no RBAC matrix.

Multiple RBAC policies might have the same RBAC matrix, for this reason the policy cannot be uniquely reconstructed from an RBAC matrix. The matrix is only a static representation of all explicit and implicit permissions, but it does not contain the original user role hierarchy.

Objects of an RBAC matrix do not need to be individual entities but could also be abstract groups. For example, in the RBAC matrix given in Table 1.1, the objects are not single documents but types of documents.

A more detailed introduction into access control can, e.g., be found in [And01; And08; Bis09].

2.4 Inference Control

Information systems challenge the security administrator with a trade-off between *availability* and *confidentiality*. That means on the one hand the system should provide useful information to a given user according to her permissions, but on the other hand it must be guaranteed that certain information remains hidden. In this thesis, we apply access control methods to protect explicit knowledge. Orthogonally, we investigate inference control methods to protect also implicit knowledge. In a traditional information system, where all the available information is stored explicitly, it is possible to simply label information items with the roles that users must have to be allowed to receive this particular information. In knowledge representation systems, this approach does not work anymore since new knowledge can be inferred. Deciding which knowledge can be given to a user without revealing secrets leads to the *inference problem* [Den82; FJ02; Bis09]: avoiding a situation where a user can infer knowledge that she should not have access to using knowledge she is allowed to access. Inference Control targets to solve the inference problem by changing a system or its behavior. The basic principle is to define a set of secrets, and deliver only answers to a user's query that do not uncover any secret. The two basic mechanisms for inference control [Bis09] are *dynamic monitoring and distortion* and *static verification and modification*.

Dynamic monitoring and distortion

Dynamic monitoring and distortion is performed while the system is running. It requires a *log*, which is maintaining the user's assumed a priori knowledge and previous answers. It further requires a *censor*, which decides whether an answer to a query would uncover any secret. If this is the case, the answer is distorted either by *refusal* or *lying*. A refusal is a notification that an answer cannot be given. A lie is an answer different from the ordinary answer which would be given. The distortion is done in a way so that the user is not able to detect a lie or exploit a refusal in order to gain knowledge she should not gain.

For ontologies, variations of this approach without a log are [BSH07] with the possibility to hide subsets of TBox and ABox, and [CS09] for query rewriting with the TBox assumed to be completely public.

For scenarios where system response time to a query is important and consequences are pre-computed, dynamic monitoring might not be feasible and instead as much as possible should be prepared before the system is started.

Static verification and modification

Static verification and modification is performed before the system is started. It inspects the original system instance to determine whether there is a permitted query sequence under which some harmful inference could occur. If so, a modification guarantees to remove such possibilities by replacing the original instance by an alternative instance, which is *inference-proof*. For a system with LBAC, the informal notion of the *inference-proof* property from [Bis09] is:

> If the label assignments permit access to entities or their relationships, then they also permit access to all inferable entities and relationships.

For example, restricting access to an ontology could be implemented by considering the axioms as objects to be protected in the sense of a LBAC model. Then an inference-proof label assignment can be formally defined as follows.

Definition 23 (Inference-proof label assignment)**.** Let a consequence c follow from a set of axioms $\{a_1, \ldots, a_n\}$. Let an LBAC model be given with $(L, \leqslant)$ a labeling lattice, each axiom a_i is labeled with ℓ_i and let a user labeled with ℓ_u be allowed to read an axiom a_i iff $\ell_u \leqslant \ell_i$. An assignment of label ℓ_c to c is *inference-proof* if $\{a_1, \ldots, a_n\} \models c \implies \ell_c \geqslant \ell_1 \otimes \ldots \otimes \ell_n$ holds.

The above definition is dual to the original definition in [Bis09] where a user labeled with ℓ_u is allowed to read an axiom a_i iff $\ell_u \geqslant \ell_i$, and so the implication is $\{a_1, \ldots, a_n\} \models c \implies \ell_c \leqslant \ell_1 \oplus \ldots \oplus \ell_n$, which is dual to the one above. Note that an inference-proof label assignment to a consequence ensures that a user can see the consequence if she can see a set of axioms from which it follows, but not the inverse. It does not say that, if she can see the consequence, then she can see a set of axioms from which it follows.

If a system instance is not inference-proof, an alternative instance could, e.g., be obtained by removing ontology axioms or by changing the label assignment. The creation of an alternative instance can be seen as a reduction of inference control to access control, since the inference problem is solved before the system is started and only access control needs to be performed at runtime.

A more detailed introduction into inference control can, e.g., be found in [Bis09].

3 Representing and Completing Role-Based Access Control Policies

In this chapter we show how to represent RBAC policies as DL ontology and how to complete RBAC policies in the sense of adding new knowledge that has neither been implicit nor explicit before.

The first section discusses why and how it makes sense to represent an RBAC policy as ontology. We discuss related work on a DL representation for an extended RBAC model and show how flaws of this representation can be fixed. The RBAC extension introduces object classes and an object class hierarchy. However, in this approach it is clear that no new knowledge is obtained that was not already contained in the RBAC policy. Basically, in addition to RBAC policy and RBAC matrix, it just adds a third representation based on Description Logics.

In the second section we introduce a systematic and non-redundant method for interviewing the security engineer in order to extend a given RBAC policy with new knowledge that has not been known before, neither implicit nor explicit. We start from an RBAC matrix, write it as formal context, interpret and transform it appropriately so that a known method from FCA called attribute exploration can be applied. A new DL representation is introduced to capture knowledge from the RBAC matrix and the newly obtained knowledge. Such knowledge comprises the role hierarchy or more specific constraints as for example "nobody is allowed to write and approve the same document." The fact, that a role hierarchy or another constraint is true for the given RBAC matrix does not mean it is true in general, which can only be decided by interviewing the security engineer. Obtaining the user role hierarchy in the form of a lattice, prepares the ground for Chapter 4, since it might be used as the basis for the labeling lattice.

In the third section we improve the method from the second section in the sense that the security engineer has to answer fewer questions. Implications which are already known a priori, e.g. from a known user role hierarchy or a known object class hierarchy, are exploited and not ignored as it is the case in the second section.

Parts of this chapter have been published in [DK09; KHD08; KH08], and a US patent has been filed under [DK10].

3.1 Representing an RBAC Policy with Object Class Hierarchy in DL

It seems natural to represent an RBAC policy by an ontology, since an RBAC policy specifies explicit permissions from which implicit permissions follow. DLs were introduced to specify explicit knowledge, from which implicit knowledge follows and can be made explicit by a reasoner.

In this section we discuss approaches from literature to represent RBAC policies by means of a DL ontology. We specifically focus on an extension of RBAC, called RBAC with object class hierarchy (RBAC-CH). The additional hierarchy of object classes makes it not only possible to inherit permissions along the user role hierarchy but also along a hierarchy of object classes. For example, permissions for marketing documents could be enforced for all kinds of special marketing documents, e.g. for flyers, brochures and success story documents.

Based on our discussion, we introduce a representation of RBAC-CH as ontology, which eliminates flaws in a proposal from related work. For a given RBAC-CH policy, the RBAC-CH matrix contains the complete set of all explicit and implicit permissions. We show how the RBAC-CH matrix can be easily computed for a given RBAC-CH ontology by just querying a reasoner. The resulting matrix helps for example the security engineer to verify the effect of user role and object class hierarchy. A comparison of the RBAC-CH matrix computed with the flawed proposal to the matrix computed with our proposal shows that the wrong results are corrected.

Figure 3.1 illustrates the workflow for representing an RBAC-CH policy including explicit permissions, user role hierarchy and object class hierarchy and computing the RBAC-CH matrix.

The result of this section is the insight that it makes sense to use DLs to represent an RBAC policy since common tasks related to RBAC policies can be reduced to DL reasoning tasks. However, it is clear that no new knowledge is obtained that was not already contained in the RBAC policy. Basically, in addition to RBAC policy and RBAC matrix, it just adds a third representation based on Description Logics.

3.1.1 Existing Approaches

Literature provides a wide range of proposals to represent an RBAC policy with DLs [Zha+05; Hei+06; CCT07; CS07b; CS07a; KHP07; BM08; Fin+08]. This is useful since common tasks related to RBAC policies can be reduced to DL reasoning tasks. This includes

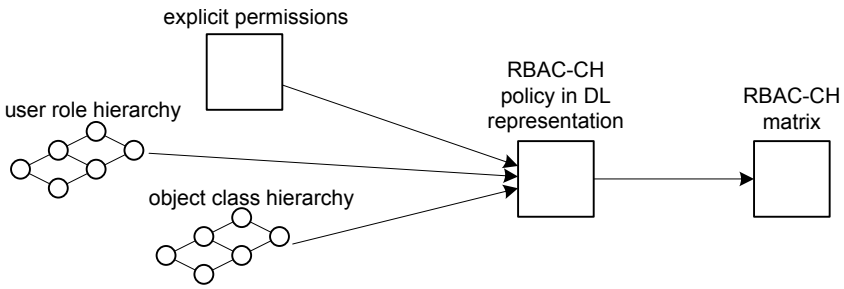

Figure 3.1: Workflow for representing an RBAC-CH policy in DL and computing the RBAC-CH matrix

- computing implicit from explicit knowledge about permissions,
- querying an access control decision,
- and checking a policy for consistency.

Modeling the knowledge of an RBAC policy in Description Logic allows us to employ existing highly-optimized reasoners to carry out those tasks.

As pointed out in the introduction to RBAC in Section 2.3, the RBAC model is in principle not restricted to control access on individual objects only, but also on groups of objects. Only [CS07b] takes classes of objects and a hierarchy among those classes into account. However, the proposed DL representation has several flaws, which are discussed in [KHD08; KH08] and are summarized here as follows.

- *Inverted object class hierarchy:* The object classes are represented by DL concepts. The object class hierarchy is represented by the subsumption hierarchy but in the opposite direction as would be expected with respect to DL semantics. The example in [CS07b] contains the GCI $File \sqsubseteq ExeFile$ which would mean that every file is an executable file, which is obviously not the case.
- *Existential quantification used inappropriately:* For example, in order to say "all system administrators can execute all files" the proposed representation introduces the GCI $SysAdmin \sqsubseteq \exists canExecute.File$, but this means "all system administrators can execute at least one file."
- *Concept assertions not used:* RBAC assigns individual users to roles and furthermore the extension classifies individual objects with object classes. DL axioms, specifically concept assertions, already offer a standard way to classify individuals from the ABox with concepts of the TBox. Instead, the object properties *assign* and *classify* are introduced. Furthermore, since object properties allow only pairs of individuals but not pairs of an individual and a concept, nominals are used to represent each object class and each user role twice: as individual and as concept.

Nevertheless, the work is relevant since it introduces the idea of object classes and an object class hierarchy. The discussed flaws motivated us to propose an improved DL representation. In the next section we give a formal definition of RBAC extended by object classes and an object class hierarchy. Based on that, we propose an improved DL representation in Section 3.1.3. The running example in [CS07b] is not correct, which might be due to the inappropriate DL representation. We compare these results with the results obtained from our representation in Section 3.1.4.

3.1.2 Extending RBAC by an Object Class Hierarchy

In this section we will give a formal definition for the extension of RBAC to RBAC-CH, presented in [CS07b].

As introduced in Section 2.3, RBAC assigns permissions not directly to users but to user roles. User permissions are indirectly determined by their assignment to user roles. RBAC-CH adds a further indirection since permissions are not defined on objects but on object classes. Permissions on objects are indirectly determined by their assignment to object classes. The formal definition of RBAC-CH is as follows.

Definition 24 (RBAC-CH)**.** The *RBAC-CH model* has the following components:

- $U, \mathcal{R}, \mathcal{A}, \mathcal{O}, \mathcal{C}$ the set of *users*, *roles*, *actions*, *objects*, *object classes* respectively
- $PA \subseteq \mathcal{R} \times A \times \mathcal{C}$ a many-to-many-to-many assignment relation for permissions of roles to perform actions on object classes
- $UA \subseteq U \times \mathcal{R}$ a many-to-many assignment relation of users to roles
- $OA \subseteq O \times \mathcal{C}$ a many-to-many assignment relation of objects to object classes
- $RH \subseteq \mathcal{R} \times \mathcal{R}$ a partial order on $\mathcal{R}$ called *user role hierarchy* also written as $\leqslant_{\mathcal{R}}$
- $CH \subseteq \mathcal{C} \times \mathcal{C}$ a partial order on $\mathcal{C}$ called *object class hierarchy* also written as $\leqslant_{\mathcal{C}}$
- implicit permissions: $r_1 \leqslant_{\mathcal{R}} r_2$ (r_1 inherits permissions from r_2) and $c_1 \leqslant_{\mathcal{C}} c_2$ (c_1 is a specialization of c_2) and $(r_2, a, c_2) \in PA \implies (r_1, a, c_1) \in PA$

A concrete instance of the RBAC-CH model, with concrete users, roles, actions, objects, object classes, etc. is called an *RBAC-CH policy*.

For a comparison to RBAC, see Definition 21. Figures 3.2 and 3.3 illustrate the difference between RBAC and RBAC-CH, where circles represent sets and arrows represent relations between sets the arrows are pointing to. It can be seen that not only the user role hierarchy, but also the object class hierarchy adds further implicit permissions to the initial set of explicit permissions.

In Section 2.3, the RBAC matrix has been introduced as a static representation of an RBAC policy containing all explicit and implicit permissions. Similarly, the *RBAC-CH matrix* can be introduced as a static representation of an RBAC-CH policy. Formally, it is defined as follows.

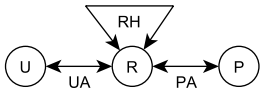

Figure 3.2: RBAC (source [SFK00])

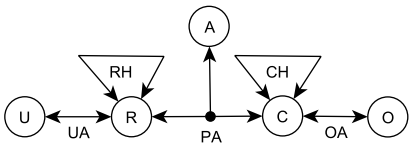

Figure 3.3: RBAC with Object Class Hierarchy

Definition 25 (RBAC-CH matrix)**.** Let M be an access control matrix with S, O, A the set of subjects, objects, actions respectively. Let furthermore PA be the permission assignment relation and $\mathcal{R}, \mathcal{C}, A'$ be the set of roles, object classes, actions respectively of an RBAC-CH policy. The matrix M is an *RBAC-CH matrix* for the RBAC-CH policy if $S = \mathcal{R}, O = \mathcal{C}, A = A'$ and $(r, a, c) \in PA \iff a \in M[r, c]$ holds.

In Section 2.3 we discussed that multiple RBAC policies can have the same RBAC matrix, and the policy cannot be reconstructed from the matrix. This is true also for RBAC-CH policy and RBAC-CH matrix.

Any RBAC-CH matrix is also an RBAC matrix. This is true because for every RBAC-CH policy there is some RBAC policy with the same set of explicit and implicit permissions. The implicit permissions resulting from the object class hierarchy in RBAC-CH can always be modeled explicit in RBAC. One could compute all implicit permissions of an RBAC-CH policy following from the object class hierarchy and express them explicitly in an RBAC policy. In this sense, RBAC-CH does not offer higher expressivity than RBAC and RBAC is more general than RBAC-CH.

3.1.3 An Improved DL Representation of RBAC-CH

The flaws in the DL representation of RBAC-CH provided by [CS07b] have been discussed in Section 3.1.1. They motivated us to provide an improved representation which is presented in this section.

A crucial modeling issue is which DL constructors are needed in order to decide which DL is used. Besides inverse properties and subproperty relationships, we need complex role inclusion axioms, i.e. axioms $P_1 \circ P_2 \sqsubseteq P$. Under certain restrictions, this particular feature has been introduced in the OWL 2 standard [MPSP09], which is based on the DL $\mathcal{SROIQ}(\mathbf{D})$ [HKS06].

Each user role and object class is represented by a concept: $UserWithRole$ is the superconcept of all user roles, *Object* is the superconcept of all object classes. Objects are instances of *Object* and its subconcepts and users are instances of $UserWithRole$ and its subconcepts. Thus technically the classification of an object with an object class and the assertion of a role to a user are represented by concept assertions. The user role hierarchy $\leqslant_{\mathcal{R}}$ and the object class hierarchy $\leqslant_{\mathcal{C}}$ are represented by the subsumption relation $\sqsubseteq$ between the respective concepts.

The definition of allowed actions for user roles on object classes has an important requirement: we want to express that for a given allowed action, *every* individual of a

user role is in relation to *every* individual of an object class. An example is "all system administrators may read all files." Obviously it is essential to express such statements in any RBAC representation, but statements like these are not generally supported by $\mathcal{SROIQ}(\mathbf{D})$. However, in [RKH08] Rudolph et al. propose the *concept product* to express such statements, which can be simulated in the DL $\mathcal{SROIQ}(\mathbf{D})$. The concept product putting all individuals of concept C in relation to all individuals of concept D by object property P is written as $C \times D \sqsubseteq P$. The following steps are performed to simulate the concept product in $\mathcal{SROIQ}(\mathbf{D})$ according to [RKH08].

- delete the axiom $C \times D \sqsubseteq P$
- add complex role inclusion $P1 \circ P2 \sqsubseteq P$, where $P1, P2$ are fresh property names
- introduce fresh nominal $\{a\}$ and add axioms $C \sqsubseteq \exists P1.\{a\}$ and $D \sqsubseteq \exists P2^{-}.\{a\}$

Definition 24 of RBAC-CH introduces the ternary permission assignment relation PA. Since the concept product is a binary relation, we split the ternary permission assignment relation into multiple binary relations, one for each allowed action. This is reasonable since the set of actions is usually small. Each allowed action is represented by an object property and the respective concept products are defined to be subproperties. In the following we give an example.

Example 11. The concept product can be applied in order to express the permission assignment "all system administrators may read all files" with the concept product $SysAdmin \times File \sqsubseteq mayRead$:

$$\begin{aligned} mayRead_1 \circ mayRead_2^{-} &\sqsubseteq mayRead \\ SysAdmin &\sqsubseteq \exists mayRead_1.\{a\} \\ File &\sqsubseteq \exists mayRead_2.\{a\} \end{aligned}$$

The introduced axioms define permissions of user roles on object classes, user role hierarchy, object class hierarchy and the assignment of users to user roles and objects to object classes. Permissions of users to objects are implicit consequences of those axioms.

The RBAC-CH matrix, containing all explicit and implicit permissions, can be queried from a reasoner as we show in the following section.

3.1.4 Computing the RBAC-CH Matrix for an RBAC-CH Policy

From the DL representation of an RBAC-CH policy described in the last section, the RBAC-CH matrix can be constructed by querying a reasoner for implicit and explicit knowledge about the RBAC-CH policy. In this section, we do this for an example scenario from [CS07b]. A comparison of our computed RBAC-CH matrix with the one from [CS07b] unveils that the latter is not correct, probably due to the inappropriate DL representation.

The considered set of user roles consists of *Remote Client* (RemCli), *Local Client* (LocCli), *Manager* (Mag), *Operating System Developer* (OSDev) and *System Administrator* (SysAdmin). The considered set of object classes consists of *System File*

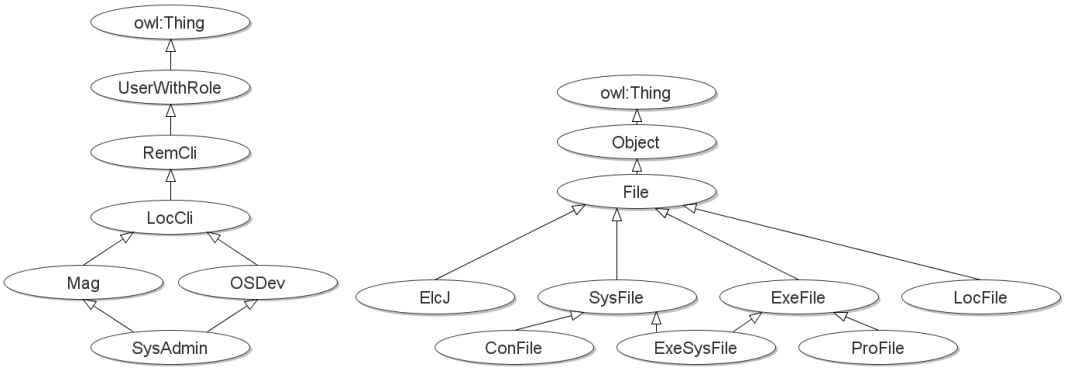

Figure 3.4: Subsumption hierarchy of user role concepts and object class concepts (arrows point from subsumee to subsumer)

	ElcJ	LocFile	ConFile	SysFile	ExeSysFile	ProFile	ExeFile	File
SysAdmin								r,w,x
Mag			r,w					
OSDev								
LocCli	r							
RemCli		r,w					x	

Table 3.1: Explicit permissions of user roles on object classes

(SysFile), *Electronic Journal* (ElcJ), *Executable File* (ExeFile), *Local File* (LocFile), *Configuration File* (ConFile), *Program File* (ProFile) and *Executable System File* (ExeSysFile). The object class hierarchy and user role hierarchy is represented as subsumption hierarchy, given in Figure 3.4.

Explicit permissions of user roles on object classes are represented by the concept product as described in the last section. Table 3.1 specifies the explicit permissions, where x represents action $canExecute$, r represents action $canRead$ and w represents action $canWrite$. The reading direction to construct the concept products is $UserWithRole \times Object \sqsubseteq action$.

Implicit permissions can be queried from a reasoner. Since the concept product is currently not supported in tools, we cannot read out implicit concept products made explicit by the reasoner directly. However, this can easily be solved technically by introducing a "representative individual" for each user role concept and object class concept and read out the allowed actions represented by object properties. From the definition of the concept product it follows that the respective action is allowed for *all* individuals of the user role concept or object class concept respectively.

The implicit permissions of the given RBAC-CH policy are given by the RBAC-CH matrix in Table 3.2. For a comparison, the incorrect implicit permissions from [CS07b] are given in braces. For example, an obvious error is that the user role $SysAdmin$ has the permission to execute every $File$, but not every $SysFile$, $ConFile$, $LocFile$ and $ElcJ$ which are subconcepts of $File$.

	ElcJ	LocFile	ConFile	SysFile	ExeSysFile	ProFile	ExeFile	File
SysAdmin	r,w,x (r)	r,w,x (r,w)	r,w,x (r,w)	r,w,x (r,w)	r,w,x	r,w,x	r,w,x	r,w,x
Mag	r	r,w	r,w		x	x	x	
OSDev	r	r,w	(r,w)	(r,w)	x (r,w,x)	x (r,w,x)	x (r,w,x)	
LocCli	r	r,w			x	x	x	
RemCli		r,w			x	x	x	

Table 3.2: RBAC-CH matrix (conflicts with [CS07b] given in parentheses)

In summary, our improved DL representation for RBAC-CH eliminates the flaws of related work discussed in Section 3.1.1. However, it still leaves room for improvement. For example, the subsumption hierarchy of user roles is not intuitive. It makes no sense in general that a system administrator is a special manager. This is only true with respect to her permissions. The new representation discussed in Section 3.2 is more sophisticated and eliminates also that problem.

3.2 Policy Completion Starting From an RBAC Matrix

The previous section showed that DLs can be used to represent an RBAC policy. However, it is clear that no new knowledge is obtained that was not already contained in the RBAC policy. Basically, in addition to RBAC policy and RBAC matrix, it just adds a third representation based on Description Logics. In this section we take the RBAC matrix representation as given. We decided for this representation, since it can be obtained from any RBAC implementation, also from old legacy systems, by reading out its static behavior. As introduced in Section 2.3, the objects in an RBAC matrix do not need to be individual entities but could also be abstract groups of entities. This is always the case with RBAC-CH matrices. In Section 3.1.2 we discussed, that RBAC is more general than RBAC-CH in the sense that one could compute all implicit permissions of an RBAC-CH policy following from the object class hierarchy and express them explicitly in an RBAC policy. In this section, we focus on RBAC matrices with objects being abstract groups of individual entities, but do not restrict to RBAC-CH.

We show how to obtain new knowledge that has neither been explicit nor implicit before. For instance, we will see in our example scenario that no user is ever allowed to write and approve the same document, which makes sense for a review process but was known before neither explicit nor implicit. We show how to obtain the new knowledge in a systematic, non-redundant way based on the attribute exploration algorithm known from FCA. Completing DL ontologies using FCA has been done, e.g., in [Baa+07a] but is not directly applicable here. Two contributions of this section allow to use attribute exploration for our purposes: (1) since the algorithm only works on two-dimensional contexts and the RBAC matrix can be interpreted as three-dimensional context, appropriate transformations are required; (2) since the algorithm further works on objects and not on classes, but the RBAC matrix specifies actions allowed for *classes* of users on *classes* of objects, appropriate interpretations of the matrix are required for allowed actions of users on objects. We introduce the

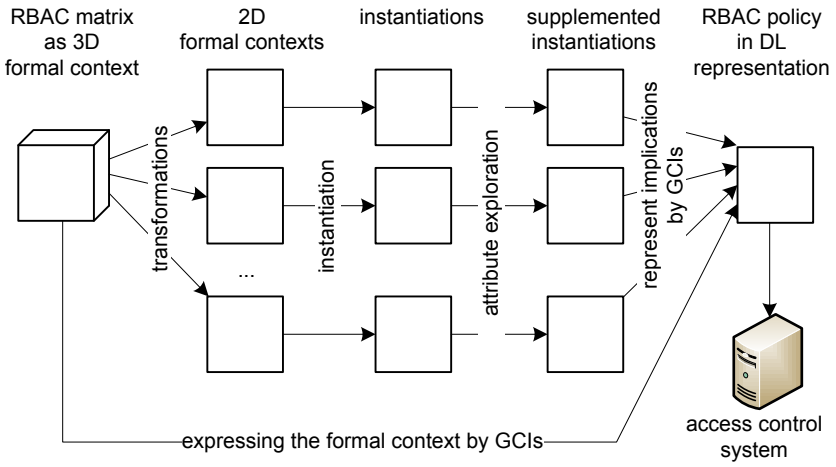

Figure 3.5: Workflow for representing and completing an RBAC policy

prohibitive, permissive, and strict interpretation of crosses in the RBAC matrix and explain that no new knowledge can be obtained for the strict interpretation while it is possible and makes sense for the other two interpretations.

Figure 3.5 illustrates the complete workflow from the RBAC matrix to an extended RBAC policy in DL representation. The resulting extended RBAC policy can be sent to an access control system in order to be enforced.

The structure is as follows: In Section 3.2.1, all relevant notions are formally defined, and the running example we will use is introduced. Moreover, in this section the three possible interpretations of an RBAC matrix are discussed. In Section 3.2.2, we show how the knowledge explicitly contained in an RBAC matrix can be expressed by means of DL GCIs. The resulting DL representation is more sophisticated than the one from Section 3.1.3 and can be extended with the newly obtained knowledge from the following two sections. In Section 3.2.3, we thoroughly discuss how attribute exploration can be used in order to obtain additional knowledge, and how it can be represented with DL GCIs. The newly obtained knowledge is added as additional constraints to the DL representation of the RBAC policy. In Section 3.2.3, we apply our approach to a real-life example.

3.2.1 The RBAC Matrix as Formal Context

In this section, all relevant notions which will be used in this section are formally defined, and our working example is introduced.

Vocabulary: As already mentioned, our starting point is a three-dimensional matrix, where the three dimensions are the *roles*, *document types* and *actions*. As defined in Section 2.3, *permissions* are tuples of action and object, i.e. document types here. As noted in the Preliminaries Chapter in Section 2.1, we avoid mixing up user roles and DL roles by using the OWL terminology "property" for a DL role. In our ongoing formalization, both roles and document types will be represented as

concept names of a (appropriately chosen) DL, and each action will be represented as a property between roles and document types. That is, we consider a DL vocabulary which consists of a set $\mathbf{R}$ of *role names*, a set $\mathbf{D}$ of *document type names*, and of a set $\mathbf{A}$ of *allowed action names*. The vocabulary of these names will be denoted $\mathbf{V}$.

We will use a working example with specific roles, document types and actions. As already introduced in our scenario in Section 1.2, we consider the actions mayApprove, mayWrite and mayRead, which are abbreviated by MA, MW and MR, respectively. The document types are user manual, marketing document, customer contract document, terms of use document, installation guide, external technical interface document, design document and rating entry, abbreviated by UM, MD, CCD, ToUD, IG, ETID, DD, RE. The roles are marketplace visitor, customer, development engineer, service vendor, legal department employee, service provider, marketing employee, technical editor and customer service employee, abbreviated by MV, CU, DE, SV, LDE, SP, ME, TE and CSE.

Formal Contexts: The three-dimensional matrix already introduced in Table 1.1 with roles, document types and actions is formalized as a triadic formal context $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}} := (\mathbf{R}, \mathbf{D}, \mathbf{A}, I)$.

Our aim is to conduct an attribute exploration in order to explore dependencies between different roles, different document types, or different actions. As attribute exploration is applied to dyadic contexts, we have do derive such contexts from the given triadic context. This can be done in several ways.

1. First, we can consider "slices" of the triadic context. For our goal, it is most useful to consider the "slice" for each $\mathsf{A} \in \mathbf{A}$. That is, for a given $\mathsf{A} \in \mathbf{A}$, we consider $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}} := (\mathbf{R}, \mathbf{D}, I^{\mathsf{A}})$, where $(\mathsf{R}, \mathsf{D}) \in I^{\mathsf{A}} :\Leftrightarrow (\mathsf{R}, \mathsf{D}, \mathsf{A}) \in I$.

2. Next, we can consider the dyadic contexts as a formal context with objects and attributes, where the set of attributes is one of the sets $\mathbf{R}$, $\mathbf{D}$, $\mathbf{A}$, and the set of objects is the cross-product of the remaining two sets. E.g. we can consider the context $\mathbb{K}_{\mathbf{R}\times\mathbf{A},\mathbf{D}} := (\mathbf{R}\times\mathbf{A}, \mathbf{D}, I^{\mathbf{R}\times\mathbf{A},\mathbf{D}})$ with $((\mathsf{R}, \mathsf{A}), \mathsf{D}) \in I^{\mathbf{R}\times\mathbf{A},\mathbf{D}} \Leftrightarrow (\mathsf{R}, \mathsf{D}, \mathsf{A}) \in I$. This is a straight-forward transformation. To simplify notations, we will denote the incidence relation again by I, thus writing $(\mathbf{R}\times\mathbf{D}, \mathbf{A}, I)$. We can construct six dyadic contexts this way, namely $\mathbb{K}_{\mathbf{R}\times\mathbf{D},\mathbf{A}}$, $\mathbb{K}_{\mathbf{A}\times\mathbf{R},\mathbf{D}}$, $\mathbb{K}_{\mathbf{D}\times\mathbf{A},\mathbf{R}}$ and the respective named variants with identical cross table $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$, $\mathbb{K}_{\mathbf{R}\times\mathbf{A},\mathbf{D}}$, $\mathbb{K}_{\mathbf{A}\times\mathbf{D},\mathbf{R}}$.

3. For a given context $\mathbb{K} := (G, M, I)$, when attribute exploration is conducted, sometimes it is sensible to add an additional attribute $\perp$ to M, which satisfies $\neg\exists g \in G : (g, \perp) \in I$. We use $\mathbb{K}_{\perp} := (G, M \cup \{\perp\}, I)$ to denote this context (again, we simply 'reuse' the symbol 'I' for the incidence relation). In our example no agent will be allowed to write and approve the same document, thus $\mathsf{mayApprove} \wedge \mathsf{mayWrite} \to \perp$.

As each of the formal contexts only deals with *names* for roles, document types, and actions, but not with instances of these names (in some DL interpretations, see below), all these formal contexts are called *$\mathcal{T}$-contexts*.

Interpretations: The DL-interpretations for RBAC matrices are straightforwardly defined: For our setting, a *DL-interpretation for* $\mathbf{V}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with a non-empty *domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ which satisfies:

Interpretation	Cross	No cross
strict	allowed for all individuals	prohibited for all individuals
permissive	allowed for all individuals	allowed for some individuals
prohibitive	allowed for some individuals	prohibited for all individuals

Table 3.3: Variants how to interpret a cross in the context

- $\mathsf{R}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each $\mathsf{R} \in \mathbf{R}$. Moreover, we set $\mathbf{R}^{\mathcal{I}} := \bigcup_{\mathsf{R} \in \mathbf{R}} \mathsf{R}^{\mathcal{I}}$.
  The elements $r \in \mathbf{R}^{\mathcal{I}}$ are called *agents*.
- $\mathsf{D}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each $\mathsf{D} \in \mathbf{D}$. Moreover, we set $\mathbf{D}^{\mathcal{I}} := \bigcup_{\mathsf{D} \in \mathbf{D}} \mathsf{D}^{\mathcal{I}}$.
  The elements $d \in \mathbf{D}^{\mathcal{I}}$ are called *documents*.
- $\mathsf{A}^{\mathcal{I}} \subseteq \mathbf{R}^{\mathcal{I}} \times \mathbf{D}^{\mathcal{I}}$ for each $\mathsf{A} \in \mathbf{A}$
- $\mathbf{R}^{\mathcal{I}} \cap \mathbf{D}^{\mathcal{I}} = \emptyset$ (nothing is both agent and document)
- $\mathbf{R}^{\mathcal{I}} \cup \mathbf{D}^{\mathcal{I}} = \Delta^{\mathcal{I}}$ (everything is either agent or document)

Note that the first two conditions are standard conditions for DL interpretations, whereas the last 3 conditions are additional constraints.

Permissive, Prohibitive and Strict Interpretations: For some applications, it might not be realistic to assume that an agent's permissions are completely defined by her assigned user roles. Instead she might have additional individual permissions and prohibitions. As each formal object and attribute of $(\mathbf{R}, \mathbf{D}, \mathbf{A}, I)$ stands in fact for a whole class of agents respectively documents, it is not a priori clear what the semantics of the incidence relation I with respect to an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is. So we have to clarify the meaning of I.

First we might assume that a relationship $(\mathsf{R}, \mathsf{D}, \mathsf{A}) \in I$ means that *each* agent $r \in \mathsf{R}^{\mathcal{I}}$ can perform action $\mathsf{A}^{\mathcal{I}}$ on *each* document $d \in \mathsf{D}^{\mathcal{I}}$. So a cross in the cross-table of the context $(\mathbf{R}, \mathbf{D}, I^{\mathsf{A}})$ grants actions allowed on documents by agents, and we can read from the context which permissions are *at least* granted to agents.

Vice versa, we might assume that a missing relationship $(\mathsf{R}, \mathsf{D}, \mathsf{A}) \notin I$ means that *no* agent $r \in \mathsf{R}^{\mathcal{I}}$ can do action $\mathsf{A}^{\mathcal{I}}$ on *any* document $d \in \mathsf{D}^{\mathcal{I}}$. So a missing cross in the cross-table of the context $(\mathbf{R}, \mathbf{D}, I^{\mathsf{A}})$ prohibits that actions are granted to agents, and we can read from the context which actions are *at most* granted to agents.

And finally, we could of course assume that both conditions hold. That is, we can read from the context which permissions are *precisely* granted to agents.

These three understandings lead to the notion of permissive, prohibitive and strict interpretations (with respect to the formal context) summarized in Table 3.3. They are formally defined as follows:

- An interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is called *permissive (with respect to* $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$*)*, and we write $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models_{+} (\mathbf{R}, \mathbf{D}, \mathbf{A}, I)$, iff for all role names $\mathsf{R} \in \mathbf{R}$, all document type names $\mathsf{D} \in \mathbf{D}$, all allowed action names $\mathsf{A} \in \mathbf{A}$ we have:

$$(\mathsf{R}, \mathsf{D}, \mathsf{A}) \in I \quad \Longrightarrow \quad \forall r \in \mathsf{R}^{\mathcal{I}}\, \forall d \in \mathsf{D}^{\mathcal{I}} : (r, d) \in \mathsf{A}^{\mathcal{I}}$$

In other words, if $(\mathsf{R}, \mathsf{D}, \mathsf{A}) \in I$, we have $\mathsf{R}^{\mathcal{I}} \times D^{\mathcal{I}} \subseteq \mathsf{A}^{\mathcal{I}}$.

- An interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is called *prohibitive (with respect to* $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$*)*, and we write $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models_{-} (\mathbf{R}, \mathbf{D}, \mathbf{A}, I)$, iff for all role names $\mathsf{R} \in \mathbf{R}$, all document type names $\mathsf{D} \in \mathbf{D}$, all allowed action names $\mathsf{A} \in \mathbf{A}$ we have:

$$(\mathsf{R}, \mathsf{D}, \mathsf{A}) \notin I \quad \Longrightarrow \quad \forall r \in \mathsf{R}^{\mathcal{I}}\, \forall d \in \mathsf{D}^{\mathcal{I}} : (r, d) \notin \mathsf{A}^{\mathcal{I}}$$

  In other words, if $(\mathsf{R}, \mathsf{D}, \mathsf{A}) \notin I$, we have $(\mathsf{R}^{\mathcal{I}} \times D^{\mathcal{I}}) \cap \mathsf{A}^{\mathcal{I}} = \emptyset$.

- An interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is called *strict (with respect to* $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$*)*, iff it is both permissive and prohibitive.

We say that we use the *permissive approach* (*prohibitive approach*, *strict approach*), if we assume that each interpretation is permissive (prohibitive, strict).

Instantiations of Contexts: As already said in the introduction, it will turn out that for running attribute exploration on the context, it is reasonable not to consider the $\mathcal{T}$*-context*, but contexts where on the side of the objects, roles are replaced by "real" users respectively document types are replaced by "real" documents. Essentially, instantiations of a context contain at least all rows of the given context, and there might be more rows, but these additional rows must be extensions of rows in the given context. These contexts are now introduced.

Let one of the contexts $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}} := (\mathbf{R}, \mathbf{D}, I^{\mathsf{A}})$ $(\mathsf{A} \in \mathbf{A})$ be given. An *instantiation* of $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}}$ is a context $(R, \mathbf{D}, J^{\mathsf{A}})$, where R is a set of agents such that

- $\forall \mathsf{R} \in \mathbf{R}\ \exists r \in R\ \forall \mathsf{D} \in \mathbf{D} : (\mathsf{R}, \mathsf{D}) \in I^{\mathsf{A}} \Leftrightarrow (r, \mathsf{D}) \in J^{\mathsf{A}}$
- $\forall r \in R\ \exists \mathsf{R} \in \mathbf{R}\ \forall \mathsf{D} \in \mathbf{D} : (\mathsf{R}, \mathsf{D}) \in I^{\mathsf{A}} \Rightarrow (r, \mathsf{D}) \in J^{\mathsf{A}}$

Such a context will be denoted $\mathbb{K}^{\mathsf{A}}_{R,\mathbf{D}}$. We define similarly the instantiations $\mathbb{K}_{R\times\mathbf{A},\mathbf{D}}$ of $\mathbb{K}_{\mathbf{R}\times\mathbf{A},\mathbf{D}}$, and $\mathbb{K}_{\mathbf{A}\times R,\mathbf{D}}$ of $\mathbb{K}_{\mathbf{A}\times\mathbf{R},\mathbf{D}}$ (where again the role names are replaced by agents), as well as the instantiations $\mathbb{K}^{\mathsf{A}}_{D,\mathbf{R}}$ of $\mathbb{K}^{\mathsf{A}}_{\mathbf{D},\mathbf{R}}$ $(\mathsf{A} \in \mathbf{A})$, $\mathbb{K}_{D\times\mathbf{A},\mathbf{R}}$ of $\mathbb{K}_{\mathbf{D}\times\mathbf{A},\mathbf{R}}$, and $\mathbb{K}_{\mathbf{A}\times D,\mathbf{R}}$ of $\mathbb{K}_{\mathbf{A}\times\mathbf{D},\mathbf{R}}$ (where now the document type names are replaced by documents). Instantiations of the contexts where the actions are the attributes, i.e. instantiations $\mathbb{K}_{D\times R,\mathbf{A}}$ of $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$ (respectively $\mathbb{K}_{R\times D,\mathbf{A}}$ of $\mathbb{K}_{\mathbf{R}\times\mathbf{D},\mathbf{A}}$) are defined similarly (where on the side of the objects, both document type names and role names are replaced by "real" documents and "real" agents, respectively).

An example for an instantiation of $\mathbb{K}^{\mathsf{mayWrite}}_{\mathbf{R},\mathbf{D}}$ is given in Table 3.4.

3.2.2 Expressing the Formal Context by GCIs

In this section, we scrutinize how the information of the context $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ can be expressed by means of DLs. Note, that we add no new knowledge here, but just represent the context as an ontology.

For the permissive approach, we have to capture the condition $\mathsf{R}^{\mathcal{I}} \times D^{\mathcal{I}} \subseteq \mathsf{A}^{\mathcal{I}}$. The left expression is a *concept product*. It cannot be expressed in $\mathcal{SHOIN}(\mathbf{D})$, which is the underlying DL of OWL DL [Bec+04]. In OWL 2, there does not exist a native language construct for the concept product either, but [RKH08] provides a

	UM	MD	CCD	ToUD	IG	ETID	DD	RE
MV								
CU								×
DE	×				×	×	×	
SV								
LDE			×	×				
SP								
ME		×						
TE	×				×	×	×	
CSE			×					

	UM	MD	CCD	ToUD	IG	ETID	DD	RE
$agent_1$								
$agent_2$								×
$agent_3$	×				×	×	×	
$agent_4$								
$agent_5$			×	×				
$agent_6$								
$agent_7$		×						
$agent_8$	×				×	×	×	
$agent_9$			×					
$agent_{10}$			×	×	×			
$agent_{11}$	×		×		×	×	×	×
$agent_{12}$		×	×					

Table 3.4: The context $\mathbb{K}^{\mathsf{mayWrite}}_{\mathbf{R},\mathbf{D}}$ and one possible instantiation.

workaround to express it in OWL 2. An alternative is the non-standard constructor $\forall C.R$ which is the set of entities which stand in relation R *to all* instances of C. It can be expressed by means of negation of relations, as $\forall C.R$ is equivalent to $\forall \neg R.\neg C$. See [LS00] for a thorough discussion of the constructor. Adding it to $\mathcal{ALC}$ still yields a decidable DL, but as this constructor is non-standard, it is not supported by common DL reasoners. However, this does not restrict practical applicability due to the duality principle discussed in the next section, which allows easier constructors. Using the constructor $\forall C.R$, the condition $\mathsf{R}^{\mathcal{I}} \times D^{\mathcal{I}} \subseteq \mathsf{A}^{\mathcal{I}}$ can be expressed with the GCIs

$$\mathsf{R} \sqsubseteq \forall \mathsf{D}.\mathsf{A} \quad (\text{i.e. } \mathsf{R} \sqsubseteq \forall \neg \mathsf{A}.\neg \mathsf{D}) \qquad \text{and} \qquad \mathsf{D} \sqsubseteq \forall \mathsf{R}.\mathsf{A}^{-1} \quad (\text{i.e. } \mathsf{D} \sqsubseteq \forall \neg \mathsf{A}^{-1}.\neg \mathsf{R})$$

For the prohibitive approach, the condition $(\mathsf{R}^{\mathcal{I}} \times \mathsf{D}^{\mathcal{I}}) \cap \mathsf{A}^{\mathcal{I}} = \emptyset$ has to be captured. This can be expressed by the two GCIs

$$\mathsf{R} \sqsubseteq \forall \mathsf{A}.\neg \mathsf{D} \qquad \text{and} \qquad \mathsf{D} \sqsubseteq \forall \mathsf{A}^{-1}.\neg \mathsf{R}$$

Note that this condition is precisely the condition for the permissive approach, when we replace each action A by its complement $\neg \mathsf{A}$. This duality principle will be discussed in the next section.

If we knew that $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ is correct, and if we know which type of approach (permissive, prohibitive, strict) we use, then we can describe the information of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ by DL GCIs. We first set $\mathsf{R}_{\text{all}} := \bigsqcup_{\mathsf{R} \in \mathbf{R}} \mathsf{R}$ and $\mathsf{D}_{\text{all}} := \bigsqcup_{\mathsf{D} \in \mathbf{D}} \mathsf{D}$. Now we define the following ontology:

$$O_0 := \{\mathsf{R}_{\text{all}} \sqsubseteq \forall \mathsf{A}.\mathsf{D}_{\text{all}}\,, \mathsf{D}_{\text{all}} \sqsubseteq \forall \mathsf{A}^{-1}.\mathsf{R}_{\text{all}} \mid \mathsf{A} \in \mathbf{A}\} \cup \{\mathsf{R}_{\text{all}} \sqsubseteq \neg \mathsf{D}_{\text{all}}\} \cup \{\mathsf{R}_{\text{all}} \sqcup \mathsf{D}_{\text{all}} \equiv \top\}$$

Obviously, a general DL-interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a DL-interpretation of $\mathbf{V}$ iff it satisfies O_0. According to the chosen approach, we can now capture the information of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ as follows:

$$\begin{aligned} O_+ &:= O_0 \cup \{\mathsf{R} \sqsubseteq \forall \neg \mathsf{A}.\neg \mathsf{D}\,,\ \mathsf{D} \sqsubseteq \forall \neg \mathsf{A}^{-1}.\neg \mathsf{R} \mid (\mathsf{R},\mathsf{D},\mathsf{A}) \in I\} \\ O_- &:= O_0 \cup \{\mathsf{R} \sqsubseteq \forall \mathsf{A}.\neg \mathsf{D}\,,\ \mathsf{D} \sqsubseteq \forall \mathsf{A}^{-1}.\neg \mathsf{R} \mid (\mathsf{R},\mathsf{D},\mathsf{A}) \notin I\} \\ O_\pm &:= O_+ \cup O_- \end{aligned}$$

Again, a DL-interpretation is obviously a permissive (prohibitive, strict) interpretation of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, if it satisfies O_+ (O_-, $O_\pm$).

3.2.3 Attribute Exploration for RBAC Matrices

In this section, we discuss how attribute exploration can be utilized in order to obtain new knowledge, neither explicit nor implicit in the RBAC matrix before. It is crucial which approach (permissive, prohibitive, strict) we use, thus we first elaborate the differences between these approaches with respect to attribute exploration. In the second and third part of this section, we go into the details of an attribute exploration for instantiations of contexts in the permissive approach.

General discussion

We first compare the permissive and the prohibitive approach. In the permissive approach, the crosses in a cross-table carry information, whereas missing crosses are not informative. In the prohibitive approach, the situation is converse: Missing crosses carry information, and crosses are not informative. Missing crosses in a relation correspond to crosses in the complement of the relation. Thus if we replace in the prohibitive approach the relations mayRead,mayWrite and mayApprove by their complements $\mathsf{mayRead}^c = \mathsf{mustNotOpen}$, $\mathsf{mayWrite}^c = \mathsf{mustNotWrite}$, $\mathsf{mayApprove}^c = \mathsf{mustNotApprove}$, we have a situation similar to the permissive approach. That is, we have the following duality principle: Any account to the permissive approach can be turned into an account to the prohibitive approach (and vice versa) by replacing each action by its complement.[1] For this reason, we do not target the prohibitive approach in this section.

We assume that the set of role names, document type names, and action names is fixed. Conducting an attribute exploration on one of the $\mathcal{T}$-contexts seems for this reason to some extent pointless, as we can only confirm implications which hold but cannot add new objects with counterexamples for implications which do not hold. So we can use attribute exploration in order to check that the information in $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ is *correct* while we cannot change $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, but this idea does not tap the full potential of attribute exploration. We assume in the following that the matrix $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ is correct, but notice that this check for correctness would have avoided the inconsistency between user role hierarchy and RBAC matrix discussed in Section 3.1).

[1] But keep in mind that switching between the permissive and prohibitive approach requires changing the underlying DL-language, including the need for non-standard constructors in the permissive approach.

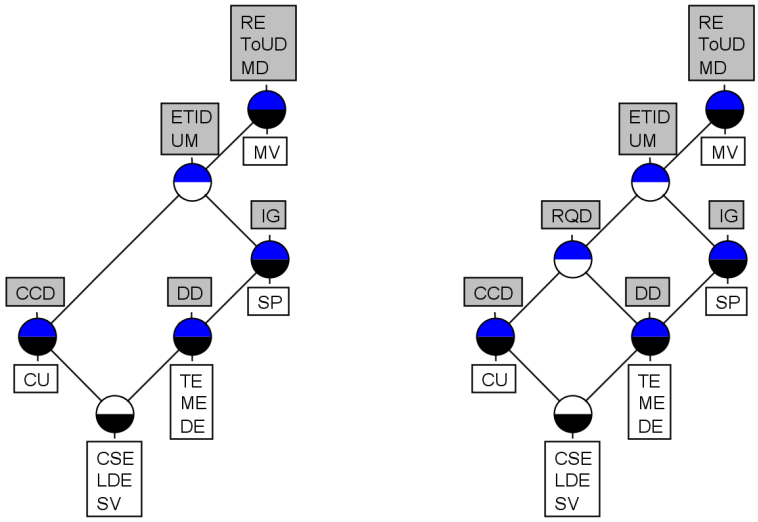

Figure 3.6: Concept lattice for context $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{R},\mathbf{D}}$ (left) and an extension (right)

Anyhow, we emphasized that in the formal context, the formal objects (the elements of $\mathbf{R}$) and attributes (the elements of $\mathbf{D}$) stand in turn for complete classes (of agents and documents). Assume we stick to the permissive approach. Assume moreover that we consider a permissive interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$. Then for a given action $\mathsf{A} \in \mathbf{A}$, agent $r \in \mathsf{R}^{\mathcal{I}}$ for a role $\mathsf{R} \in \mathbf{R}$, and document $d \in \mathsf{D}^{\mathcal{I}}$ for a document type $\mathsf{D} \in \mathbf{D}$, we might have that r can do A on d (i.e., $(r,d) \in \mathsf{A}^{\mathcal{I}}$), though we do not have $(\mathsf{R},\mathsf{D},\mathsf{A}) \in I$. That is, it is sensible to run an attribute exploration on the *instantiations* of the $\mathcal{T}$-contexts. As we will see in the next section, with attribute exploration we can in fact infer constraints for the dependencies between roles, document types and actions which are not captured by $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$.

In the strict approach, the situation is different. If we consider a strict interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, then for a given action $\mathsf{A} \in \mathbf{A}$, agent $r \in \mathsf{R}^{\mathcal{I}}$ and document $d \in \mathsf{D}^{\mathcal{I}}$, we have $(r,d) \in \mathsf{A}^{\mathcal{I}} \Leftrightarrow (\mathsf{R},\mathsf{D},\mathsf{A}) \in I$. That is, based on the given assumption that the sets of roles, document types and actions are fixed, all possible constraints for the dependencies between these entities are already captured by $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$. This observation has two consequences: First, no DL representation of the strict approach can extend the information of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, i.e., a DL formalization of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ is somewhat pointless. Second, the instantiations of $\mathcal{T}$-context are nothing but the $\mathcal{T}$-context themselves (instantiations might duplicate some rows, but this is of course of no interest), thus conducting attribute exploration in the strict approach is pointless as well. The concept lattice of the context $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{R},\mathbf{D}}$ is given in Figure 3.6 (left). Since any instantiation of $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{R},\mathbf{D}}$ equals $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{R},\mathbf{D}}$ modulo renaming objects, any instantiation has the same concept lattice. Extending $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{R},\mathbf{D}}$ with a new document type requirements document (RQD) which may be read by DE,ME,TE,SV,LDE,CSE,CU yields the concept lattice in Figure 3.6 (right).

To summarize: As the permissive and prohibitive approach are mutually dual, and as addressing the strict approach with DLs or attribute exploration is pointless, it is sufficient that we here address only the permissive approach in the following.

Attribute Exploration for Instantiations of $\mathcal{T}$-contexts

In the last part we argued why we will run attribute exploration on instantiations of $\mathcal{T}$-contexts. Before doing so, we first have to discuss how implications in $\mathcal{T}$-contexts and their instantiations are read, and then we will scrutinize some peculiarities for applying attribute exploration in our setting. In fact, due to the fact that the objects and attributes of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ stand for whole classes, the existing approaches for conducting attribute explorations on triadic contexts (e.g., [GO04]) cannot be applied to our framework.

Reading Implications We consider the two contexts of Table 3.4. In both contexts, terms of use document $\rightarrow$ customer contract document holds. For the $\mathcal{T}$-context $\mathbb{K}_{\mathbf{R},\mathbf{D}}^{\mathsf{mayWrite}}$, the objects are classes, thus this implication is read as follows:

> **$\mathcal{T}$-reading:** For each role we have that whenever every agent of that role may write all terms of use documents, then every agent of that role may write all customer contract documents as well.

For the instantiation $\mathbb{K}_{R,\mathbf{D}}^{\mathsf{mayWrite}}$ of $\mathbb{K}_{\mathbf{R},\mathbf{D}}^{\mathsf{mayWrite}}$, the objects are now instances instead of classes, thus we have a different reading of the implication. It is:

> **$\mathcal{I}$-reading:** For every agent we have that whenever she may write all terms of use documents, then the she may write all customer contract documents as well.

Implications like this cannot be read from any $\mathcal{T}$-context, thus running attribute exploration on instantiations can indeed be used to obtain new knowledge.

Please note that none of the above readings conforms to the concept inclusion terms of use document $\sqsubseteq$ customer contract document. This is due to in both implications we quantify over *all* terms of use documents and *all* customer contract documents. For the latter reading, we now show how it is correctly translated into a GCI. The implication means that for any permissive interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, we have that $\forall r \in \mathsf{R}^{\mathcal{I}} : (\forall d \in \mathsf{ToUD}^{\mathcal{I}} : (r,d) \in \mathsf{MW}^{\mathcal{I}} \rightarrow \forall d \in \mathsf{CCD}^{\mathcal{I}} : (r,d) \in \mathsf{MW}^{\mathcal{I}})$ holds. This condition is now transformed into a GCI as follows:

$$\begin{aligned} & \forall r \in \mathsf{R}^{\mathcal{I}} : \big(\forall d \in \mathsf{ToUD}^{\mathcal{I}} : (r,d) \in \mathsf{MW}^{\mathcal{I}} \rightarrow \forall d \in \mathsf{CCD}^{\mathcal{I}} : (r,d) \in \mathsf{MW}^{\mathcal{I}}\big) \\ \iff \quad & \forall r \in \mathsf{R}^{\mathcal{I}} : \big(r \in (\forall \mathsf{ToUD.MW})^{\mathcal{I}} \rightarrow r \in (\forall \mathsf{CCD.MW})^{\mathcal{I}}\big) \\ \iff \quad & (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models \forall \mathsf{ToUD.MW} \sqsubseteq \forall \mathsf{CCD.MW} \end{aligned}$$

(we have to emphasize that the direction "$\rightarrow$" of the last equivalence is only valid if we assume that $dom(\mathsf{MW}^{\mathcal{I}}) \subseteq \mathsf{R}^{\mathcal{I}}$ holds, but we assume that our interpretation satisfies O_0, which expresses this additional condition).

In general, any implication of the form $\mathsf{D}_1 \wedge \ldots \wedge \mathsf{D}_{n-1} \rightarrow \mathsf{D}_n$ in an instantiation of one of the contexts $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}}$ can be translated into the following GCI:

$$\forall \mathsf{D}_1.\mathsf{A} \sqcap \ldots \sqcap \forall \mathsf{D}_{n-1}.\mathsf{A} \sqsubseteq \forall \mathsf{D}_n.\mathsf{A}$$

Similarly, any implication of the form $\mathsf{R}_1 \wedge \ldots \wedge \mathsf{R}_{n-1} \rightarrow \mathsf{R}_n$ in an instantiation of one of the contexts $\mathbb{K}^{\mathsf{A}}_{\mathbf{D},\mathbf{R}}$ can be translated into the following GCI:

$$\forall \mathsf{R}_1.\mathsf{A}^- \sqcap \ldots \sqcap \forall \mathsf{R}_{n-1}.\mathsf{A}^- \sqsubseteq \forall \mathsf{R}_n.\mathsf{A}^-$$

If we consider an instantiation of a context where the attributes of the context are neither document type names nor role names, but instead action names, the situation is different, as now the attributes do not stand for classes of instances, but for properties between instances. In Section 3.2.4, we consider a context $\mathbb{K}_{D\times R,\mathbf{A}}$. In this context, $\mathsf{mayWrite} \rightarrow \mathsf{mayRead}$ holds. The reading of this implication is

> Whenever some agent has the permission to write some document, then this agent may read this document as well.

So we see that in this case, the implication can be translated to a simple inclusion axiom between properties, namely $\mathsf{mayWrite} \sqsubseteq \mathsf{mayRead}$.

Conducting Attribute Exploration on Instantiations

We consider the instantiation of a $\mathcal{T}$-context, where we want to run attribute exploration on. Obviously, for any $\mathcal{T}$-context $\mathbb{K}$, there exists a smallest instantiation $\mathbb{K}_{\min}$, which is isomorphic to $\mathbb{K}$, and a largest instantiation $\mathbb{K}_{\max}$. The basic idea is that we start the attribute exploration with $\mathbb{K}_{\min}$, and for implications which do not hold, we add (as usual) counterexamples to the context, until we finally reach a context $\mathbb{K}_{\text{ae}}$. Anyhow, in this process, we cannot add counterexamples in an arbitrary manner, as the context $\mathbb{K}_{\text{ae}}$ we obtain must still be an instantiation. The question is how this additional constraint can be captured by attribute exploration. First of all, we trivially have the following subset relations between the implications which hold in the contexts:

$$Imp(\mathbb{K}_{\max}) \subseteq Imp(\mathbb{K}_{\text{ae}}) \subseteq Imp(\mathbb{K}_{\min})$$

So if we run an attribute exploration on $Imp(\mathbb{K}_{\min})$, we could use $Imp(\mathbb{K}_{\max})$ as a set of additional background implications [Stu96]. Anyhow, a closer observation yields that $Imp(\mathbb{K}_{\max})$ only contains all implications of the form $\emptyset \rightarrow m$, where m is an attribute of $\mathbb{K}_{\min}$ which applies to all objects. This can easily be seen as follows: Let $\mathbb{K}_{\min} := (G_{\min}, M, I_{\min})$, let $\mathbb{K}_{\max} := (G_{\max}, M, I_{\max})$, let $M_1 := \{m \in M \mid \forall g \in G_{\min} : (g,m) \in I_{\min}\}$ and $M_2 := M - M_1$ be the complement of M_1. First of all, we obviously have for each $m_1 \in M_1$ that $\emptyset \rightarrow m_1$ holds in $\mathbb{K}_{\min}$, thus in $\mathbb{K}_{\max}$ as well. Now let $m_2 \in M_2$. Then there exists an object $g \in G_{\max}$ with $(g,m) \in I_{\max} \Leftrightarrow m \neq m_2$ for all $m \in M$. That is, there cannot exist any (nontrivial) implication in $Imp(\mathbb{K}_{\max})$ with m_2 in its conclusion.

Choice of Instantiation Contexts for Attribute Exploration

Theoretically, we could conduct an attribute exploration on the minimal instantiation of $\mathbb{K}_{\mathbf{R}\times\mathbf{A},\mathbf{D}}$. Anyhow, we observe that any instantiation of $\mathbb{K}_{\mathbf{R}\times\mathbf{A},\mathbf{D}}$ is the subposition of instantiations of the contexts $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}}$, $\mathsf{A} \in \mathbf{A}$. Generally, for any contexts $\mathbb{K}_1, \ldots, \mathbb{K}_n$ with identical attribute sets, an implication holds in each context $\mathbb{K}_1, \ldots, \mathbb{K}_n$ if and only if it holds in the subposition of these contexts. Thus if the security engineer runs an attribute exploration on the minimal instantiation of all contexts $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}}$, $\mathsf{A} \in \mathbf{A}$, there is no need to run an attribute exploration on the minimal instantiation of $\mathbb{K}_{\mathbf{R}\times\mathbf{A},\mathbf{D}}$.

The discussion above applies to the context $\mathbb{K}_{\mathbf{D}\times\mathbf{A},\mathbf{R}}$ as well. To summarize: For a *complete* investigation of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, it is sufficient that the security engineer runs an attribute exploration on the minimal instantiations of the following contexts:

- $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}}$ for each action $\mathsf{A} \in \mathbf{A}$ to infer document implications
- $\mathbb{K}^{\mathsf{A}}_{\mathbf{D},\mathbf{R}}$ for each action $\mathsf{A} \in \mathbf{A}$ to infer role implications
- $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$ to infer action implications

For the context $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$, one could add the additional attribute $\bot$ in order to obtain constraints which express the disjointness of some actions.

3.2.4 Evaluation of the Approach for a Real-Life-Example

In this section, we apply our approach to the example RBAC matrix introduced in Table 1.1. We do not conduct a complete attribute exploration: Instead we consider only the contexts $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$ and $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{D},\mathbf{R}}$.

Attribute exploration for $\mathbb{K}_{D\times R,\mathbf{A}}$

In this section, we conduct the attribute exploration on the minimal instantiation $\mathbb{K}_{\min}$ of $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$. For this exploration, as discussed in Section 3.2.1, we added an additional attribute $\bot$ to the set of attributes. An excerpt of $\mathbb{K}_{\min}$, together with its concept lattice, is provided in Figure 3.7. We observe that $Imp(\mathbb{K}_{\min})$ does not contain implications of the form $\emptyset \to m$, where m is an attribute of $\mathbb{K}_{\min}$ which applies to all objects, for this reason there are no GCIs we can already add to our ontology prior to attribute exploration.

The security engineer starts the attribute exploration on $\mathbb{K}_{D\times R,\mathbf{A}}$, thus on $\mathbb{K}_{\min}$, which has the following implications:

1. $\mathsf{MW} \to \mathsf{MR}$
2. $\mathsf{MA} \to \mathsf{MR}$
3. $\bot \to \mathsf{MR} \wedge \mathsf{MW} \wedge \mathsf{MA}$
4. $\mathsf{MR} \wedge \mathsf{MW} \wedge \mathsf{MA} \to \bot$.

	MR	MW	MA	⊥
$(mv1, um1)$				
$(cu1, um1)$	×			
$(de1, um1)$	×	×		
...				
$(sv1, ig1)$	×		×	
...				
$(cse1, re1)$	×			

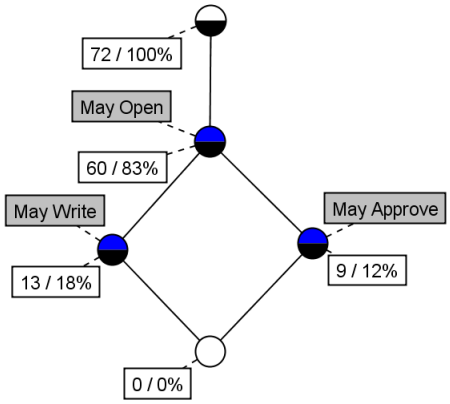

Figure 3.7: The instantiation context $\mathbb{K}_{D\times R,\mathbf{A}}$ and its concept lattice

The first implication is read: Whenever some agent can write some document, then this agent can read this document as well. It can easily be verified that this implication should indeed hold in any interpretation $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, so we add the property inclusion $\mathsf{mayWrite} \sqsubseteq \mathsf{mayRead}$ to our DL ontology. This is the first example of a statement which can be expressed with a DL statement, but not with matrix $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ alone.

The next implication can be handled analogously, and for this reason we add the inclusion $\mathsf{mayApprove} \sqsubseteq \mathsf{mayRead}$ to the ontology.

The third implication trivially holds due to the definition of $\bot$.

The last implication can, due to the first two implications, be simplified to $\mathsf{MW} \wedge \mathsf{MA} \to \bot$. Due to the definition of $\bot$, this is read: No agent can both write and approve some document. Again, the engineer decides that this implication is valid. Thus she adds the disjoint property axiom $\mathsf{MW} \sqcap \mathsf{MA} \sqsubseteq \bot$ to the ontology.

If it is later verified that the complete RBAC policy is consistent, which can be done with a DL reasoner, then each document which can be written or can be approved has to be readable and furthermore no document can be written and approved by the same agent. These are constraints which have not been contained in the matrix but were derived by our methodology.

Attribute Exploration for $\mathbb{K}^{\mathsf{mayRead}}_{D,\mathbf{R}}$

For a second example, attribute exploration is performed on the minimal instantiation context $\mathbb{K}_{\min}$ of $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{D},\mathbf{R}}$. The context $\mathbb{K}_{\min}$ looks like the left third of the cross table in Table 1.1 despite that it is transposed and document types are replaced by documents (columns are roles, rows are documents). We do not conduct a complete attribute exploration on $\mathbb{K}_{\min}$, but only provide an example for a valid and an invalid implication.

Let us first note that in $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{D},\mathbf{R}}$, the attributes SV, LDE and CSE apply to all objects. So, according to the discussion the implications $\emptyset \to \mathsf{SV}$, $\emptyset \to \mathsf{LDE}$ and

$\emptyset \to \mathsf{CSE}$ hold in all instantiations of $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{D},\mathbf{R}}$, thus we can add the GCIs $\top \sqsubseteq \forall \mathsf{SV.mayRead}^-$, $\top \sqsubseteq \forall \mathsf{LDE.mayRead}^-$ and $\top \sqsubseteq \forall \mathsf{CSE.mayRead}^-$ to our ontology, before conducting attribute exploration.

An example for an implication of $\mathbb{K}_{\min}$ is $\mathsf{TE} \to \mathsf{ME}$. During the attribute exploration, the security engineer has to decide whether this implication holds in all desired interpretations of $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{D},\mathbf{R}}$. In fact, there might be a contract document in preparation by a technical editor which is not allowed to be read by a marketing employee. Thus the security engineer adds a counterexample to the context $(\mathsf{CCD\_in\_prep}, \mathsf{TE}, \mathsf{MR}) \in I$ and $(\mathsf{CCD\_in\_prep}, \mathsf{ME}, \mathsf{MR}) \notin I$.

Another example for an implication of $\mathbb{K}_{\min}$ is $\mathsf{MV} \to \mathsf{CU}$. In fact, the security engineer realizes that this implication must hold: Any document which can be read by a marketplace visitor can be read by a customer as well. So she adds the GCI $\forall \mathsf{MV.mayRead}^- \sqsubseteq \forall \mathsf{CU.mayRead}^-$ to the ontology. This is again an example which cannot be derived from $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ alone.

3.3 Reusing Role and Object Class Hierarchy at Policy Completion

In Section 3.1 we defined RBAC-CH as an extension of RBAC. While the role hierarchy of an RBAC policy and for RBAC-CH in addition to that also the object class hierarchy can be seen as a set of implications, they have been ignored in Section 3.2. In this section, we take them into account and show that the systematic and non-redundant procedure to complete an RBAC policy from Section 3.2.3 can be made shorter in the sense that the security engineer has to answer fewer questions.

During attribute exploration, it would be cumbersome for the security engineer if she had to confirm implications that are already known a priori. An alternative is taking a set of background implications into account at attribute exploration [Stu96]. In Section 3.2.3, we already discussed, that we could use $Imp(\mathbb{K}_{\max})$ as a set of background implications during attribute exploration. We discussed that this is possible but of not much help, since $Imp(\mathbb{K}_{\max})$ only contains all implications of the form $\emptyset \to m$, where m is an attribute of $\mathbb{K}_{\min}$ which applies to all objects. The sets of background implications discussed in this section are more helpful.

Figure 3.8 extends the workflow from Figure 3.5 so that object class hierarchy and role hierarchy are taken into account at attribute exploration.

Let B_D, B_R, B_A be disjoint sets of background implications between document types, user roles and actions, respectively. For example, from the role hierarchy $(\mathcal{R}, \leqslant_{\mathcal{R}})$ and object class hierarchy $(\mathcal{C}, \leqslant_{\mathcal{C}})$ defined in Section 3.1.2, we can obtain $B_R := \{r_1 \to r_2 \mid r_2 \leqslant_{\mathcal{R}} r_1\}$, and $B_D := \{d_1 \to d_2 \mid d_2 \leqslant_{\mathcal{C}} d_1\}$.

Before attribute exploration is started, every background implication can be translated into a GCI and added to the ontology as discussed above. A preparation step prior to attribute exploration is the computation of the complete RBAC matrix as described in Section 3.1, which contains all explicit and implicit permissions of a given RBAC policy in the presence of role hierarchy and object class hierarchy. As already introduced in Section 3.2, this matrix is formalized as triadic formal context $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$ and

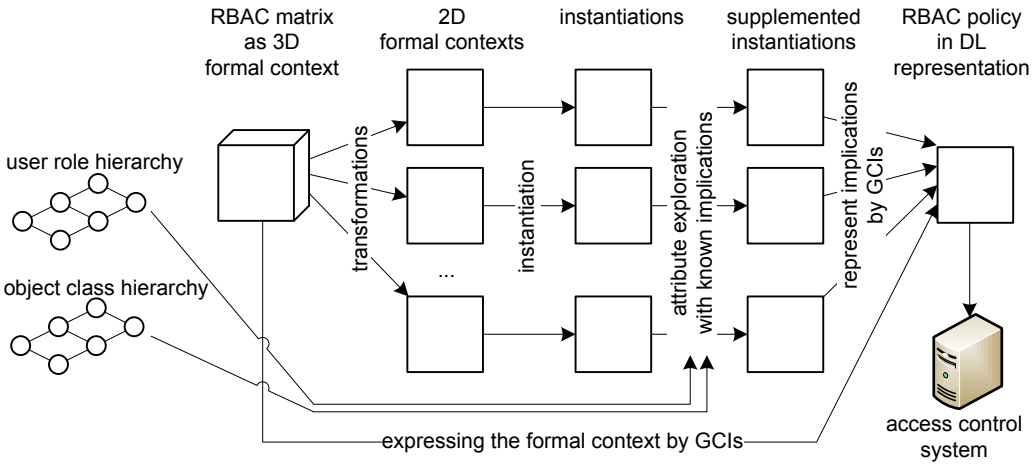

Figure 3.8: Workflow for representing and completing an RBAC policy with the support by known implications

attribute exploration is performed on minimal instantiations of that context. Similarly to the list from Section 3.2.3 for a *complete* investigation of $\mathbb{K}_{\mathbf{R},\mathbf{D},\mathbf{A}}$, it is sufficient that the security engineer runs an attribute exploration on the minimal instantiations of the following contexts, and additionally takes the respective set of background implications into account:

- $\mathbb{K}^{\mathsf{A}}_{\mathbf{R},\mathbf{D}}$ and background implications B_D for each action $\mathsf{A} \in \mathbf{A}$ to infer document implications
- $\mathbb{K}^{\mathsf{A}}_{\mathbf{D},\mathbf{R}}$ and background implications B_R for each action $\mathsf{A} \in \mathbf{A}$ to infer role implications
- $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$ and background implications B_A to infer action implications

We will now extend the attribute explorations for the real-life example from Section 3.2.4 by background implications and discuss the differences.

We start with attribute exploration on the minimal instantiation $\mathbb{K}_{\min}$ of $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{D},\mathbf{R}}$ with the background implication set $B_R = \{\mathsf{MV} \to \mathsf{CU} \wedge \mathsf{DE} \wedge \mathsf{SV} \wedge \mathsf{LDE} \wedge \mathsf{SP} \wedge \mathsf{ME} \wedge \mathsf{TE} \wedge \mathsf{CSE}\}$. The set contains one implication saying that whenever a document can be read by a marketplace visitor, then a user with any other known user role (from the fixed set of user roles) can read it as well. Thus, we can add the GCI $\forall \mathsf{MV}.\mathsf{mayRead}^- \sqsubseteq \forall \mathsf{CU}.\mathsf{mayRead}^- \sqcap \ldots \sqcap \forall \mathsf{CSE}.\mathsf{mayRead}^-$ to our ontology, before conducting attribute exploration. During attribute exploration, the implication $\mathsf{MV} \to \mathsf{CU}$ of $\mathbb{K}_{\min}$, discussed above still holds. Yet, the security engineer does not need to decide whether this implication holds, since it can already be confirmed from the set of background implications.

In a second example, we conduct attribute exploration on the minimal instantiation $\mathbb{K}_{\min}$ of $\mathbb{K}_{\mathbf{D}\times\mathbf{R},\mathbf{A}}$ with the background implication set $B_A = \{\mathsf{MA} \to \mathsf{MR}, \mathsf{MW} \wedge \mathsf{MA} \to$

$\bot\}$. Again each of the two background implications can be represented by the respective GCI and added to our ontology, prior to attribute exploration. At attribute exploration, the security engineer only has to decide about two of the above listed four implications, namely

1. $\mathsf{MW} \rightarrow \mathsf{MR}$
2. $\bot \rightarrow \mathsf{MR} \wedge \mathsf{MW} \wedge \mathsf{MA}$.

The two remaining implications discussed above, still hold in this context (since it is the same context) but can already be answered from the set of background implications so again the security engineer does not need to answer unnecessary questions.

Similarly, attribute exploration can be performed on the minimal instantiation $\mathbb{K}_{\min}$ of $\mathbb{K}^{\mathsf{mayRead}}_{\mathbf{R},\mathbf{D}}$ with a background implication set B_D.

3.4 Conclusions of the Chapter

RBAC is a standardized and well established access control model, abstracting from individual users by user roles. We have shown how to represent RBAC policies as DL ontology and how to complete RBAC policies in the sense of adding new knowledge that has been neither explicit nor implicit before.

The main argument to use a knowledge representation formalism to represent an RBAC policy is that a policy specifies explicit permissions from which implicit permissions follow. We discussed approaches from literature to represent RBAC policies by means of a DL ontology and focused specifically on an extension of RBAC, called RBAC-CH, adding a hierarchy of object classes to RBAC. We introduced a representation of RBAC-CH that has several advantages over the proposal in related work. Furthermore, we have shown that the RBAC matrix can be computed from an RBAC policy by just plugging in a reasoner and querying for all implicit permissions. The resulting matrix helps, e.g., the security engineer to verify the effect of user role and object class hierarchy. However, we pointed out that a DL representation does not add any new knowledge that was not already contained in the RBAC policy. Basically, in addition to RBAC policy and RBAC matrix, it just adds a third representation based on Description Logics.

We have shown how to complete RBAC policies in the sense of adding new knowledge that has not yet been represented in a computer, but was known to an expert, e.g. the security engineer, before. Our methodology is based on a known method from FCA, called attribute exploration. An RBAC matrix can be written as three-dimensional formal context, but needs to be interpreted and transformed appropriately, so that attribute exploration for two-dimensional contexts can be applied. In one of the discussed interpretations, the strict interpretation, the set of permissions an individual can have is strictly defined by the RBAC matrix. No additional constraints need to be checked. Under a permissive (respectively prohibitive) interpretation, individuals have additional (respectively fewer) permissions than given for their user roles. This means the RBAC matrix is no complete definition of permissions for an individual. In that case we might nevertheless have general rules and constraints which have to be

fulfilled in any case on the level of individual users. For example, nobody might be allowed to write and approve the same document for a proper review process. Rules like this on the level of individual users and individual documents can be identified by our systematic and non-redundant method based on attribute exploration. We have shown that known implications from a user role hierarchy or object class hierarchy can be exploited in order to pose fewer questions to the security engineer. We introduced a new DL representation to capture knowledge from the RBAC matrix as well as the newly obtained knowledge.

Revisiting the research questions from Section 1.4, these results answer the questions 7, 8 and 9.

Our results on the representation and completion of RBAC policies laid the ground for the next chapters on access restrictions to an ontology's axioms, since some of the results allow to reuse a given RBAC policy to prepare a labeling lattice which is required in our framework for access restrictions to an ontology's axioms and consequences.

4 Access Restrictions to Explicit and Implicit Knowledge

The framework developed in this chapter can deal with scenarios where selected sub-ontologies of a large ontology are offered as views to users, based on criteria like the user's access right, the trust level required by the application, or the level of detail requested by the user.

Instead of materializing a large number of different sub-ontologies, we propose to keep just one ontology, but equip each axiom with an annotation in the form of a label from an appropriate labeling lattice. The access right, required trust level, etc. is then also represented by a label (called user label) from this lattice, and the corresponding sub-ontology is determined by comparing this label with the axiom labels.

For large-scale ontologies, certain consequences (like the concept hierarchy) are often pre-computed. Instead of precomputing these consequences for every possible sub-ontology, our approach computes just one label for each consequence such that a comparison of the user label with the consequence label determines whether the consequence follows from the corresponding sub-ontology or not.

From a security administrator's perspective it might be necessary to change a consequence label to meet a certain specification. This can only be done by repairing the labeling of the explicit axioms.

Parts of the chapter have been published in [BKP09a; BKP09b; KP10a; KP10b].

4.1 Access Restrictions to Explicit Knowledge

In this section we equip each axiom with a label representing an access restriction. The user access right, her required trust level, etc. is then also represented by a label (called user label) from this lattice, and her visible sub-ontology is determined by comparing this label with the axiom labels.

Assume that you have a large ontology O, but you want to offer different users different views on this ontology, i.e., each user can see only a subset of the actual ontology, which is selected by an appropriate criterion. This criterion could be the user role that this user has, the level of trust in the axioms of the ontology that the user requires, the degree of certainty that the user requires, the level of details that is deemed to be appropriate for this user, etc. With user roles, each axiom label defines the user roles able to see the axiom and each user label defines the sub-ontology containing the axioms visible to this user. In the presence of trust restrictions, the user label specifies the trust level required for the ontology axiom. This supports scenarios with axioms from different sources, like company-internal with high trust level and public Web with low trust level. In the presence of uncertainty, e.g. in possibilistic reasoning, each axiom has an associated certainty degree in the interval $[0, 1]$. The user label then specifies the certainty degree required for the axioms and the consequences. Similarly, granularity restrictions (i.e., on how much details the ontology should provide for the user) can be expressed by a total order.

In principle, you could explicitly create a sub-ontology for each (type of) user, but then you might end up with exponentially many different ontologies, where each is a subset of O. Instead, we propose to keep just the large ontology O, but label the axioms in O such that a comparison of the axiom label with the user label determines whether the axiom belongs to the sub-ontology for this user or not. To be more precise, we use a labeling lattice $(L, \leqslant)$, i.e., a set of labels L together with a partial order $\leqslant$ on these labels such that a finite set of labels always has a join (supremum, least upper bound) and a meet (infimum, greatest lower bound) w.r.t. $\leqslant$.[1]. All axioms $a \in O$ are now assumed to have a label $\mathsf{lab}(a) \in L$, and the user also receives a label $\ell \in L$ (which can be read as access right, required level of trust, etc.). The sub-ontology that a user with label ℓ can see is then defined to be[2]

$$O_{\geqslant \ell} := \{a \in O \mid \mathsf{lab}(a) \geqslant \ell\}.$$

Example 12. We continue Example 1. A development engineer has user label ℓ_3, so she can see $O_{\geqslant \ell_3} := \{a_1, a_2, a_3, a_4\}$.

The Oracle Corporation provides a database technology called *Virtual Private Database* where labels are assigned to individual tuples. Access to a tuple is only allowed if the user's label dominates the tuple's label [Bis09]. Similarly our concept could be named *Virtual Private Ontology*.

4.2 Access Restrictions to Implicit Knowledge

Intuitively, for a given user all those consequences should visible, which can be inferred from her visible axioms. This can be assured by appropriate consequence labels. The consequence labels are determined by the labels of axioms that entail the consequence.

[1] We introduced a small lattice already in Figure 1.3.

[2] To define this sub-ontology, an arbitrary partial order would be sufficient. However, the existence of suprema and infima will be important for the computation of a boundary of a consequence (see below).

We determine under which restrictions on the user and axiom labels such consequence labels (called boundaries) always exist, and describe different black-box approaches for computing boundaries. Black-box means that, rather than requiring modifications of existing reasoning procedures, these approaches can use such procedures directly as sub-procedures, which allows us to employ existing highly-optimized reasoners.

Of course, the user of an ontology should not only be able to see her axioms, but also the consequences of these axioms. Thus, a user with label ℓ should be able to see all the consequences of $O_{\geqslant \ell}$. For large ontologies, certain relevant consequences are often pre-computed. The goal of the pre-computation is that certain user queries can be answered by a simple look-up in the pre-computed consequences, and thus do not require expensive reasoning during the deployment phase of the ontology. For example, in the version of the large medical ontology SNOMED CT that is distributed to hospitals, all the subsumption relationships between the concept names occurring in the ontology are pre-computed. For a labeled ontology as introduced above, it is not enough to pre-compute the relevant consequences of O. In fact, if the relevant consequence c follows from O, then we also need to know for which user labels ℓ it still follows from $O_{\geqslant \ell}$. Otherwise, if a user with label ℓ asks whether c holds, the system could not simply look this up in the pre-computed consequences, but would need to compute the answer on-the-fly by reasoning over the sub-ontology $O_{\geqslant \ell}$. Our solution to this problem is to compute a so-called *boundary* for the consequence c, i.e., an element μ_c of L such that c follows from $O_{\geqslant \ell}$ iff $\ell \leqslant \mu_c$.

As introduced in Section 2.1, there are basically two approaches for computing a boundary. The *glass-box approach* takes a specific reasoner (or reasoning technique) for an ontology language and modifies it such that it can compute a boundary. Examples for the application of the glass-box approach to specific instances of the problem of computing a boundary are tableau-based approaches for reasoning in possibilistic Description Logics [QP08; Les+08] (where the lattice is the interval $[0, 1]$ with the usual order), glass-box approaches to axiom pinpointing in Description Logics [SC03; Mey+06; Kal+05; BP10b; BP08] (where the lattice consists of (equivalence classes of) monotone Boolean formulae with implication as order [BP08]) and RDFS reasoning over labeled triples with modified inference rules for access control and provenance tracking [JF06; Flo+09]. The problem with glass-box approaches is that they have to be developed for every ontology language and reasoning technique anew and that optimizations of the original reasoning technique do not always apply to the modified reasoners. In contrast, the *black-box approach* can re-use existing optimized reasoners without modifications, and it can be applied to arbitrary ontology languages: one just needs to plug in a reasoner for this language.

The database community has been investigating a similar problem, by annotating every tuple of the database relations with an element from a commutative semiring $(K, +, \cdot, 0, 1)$ in order to represent ordered criteria, as for example data provenance (also called lineage or pedigree), trust scores, access control levels and certainty levels. For example, the data provenance instance of the problem aims at being able to tell which database tuples contributed to a given query result tuple. Since a selected relational algebra is extended in order to deal with the annotations, the approach presented in [Tan10; GKT07] can be considered a glass-box approach. As introduced

in Section 2.2, a lattice can be defined as algebraic structure $(L, \oplus, \otimes)$ and a bounded lattice is an algebraic structure $(L, \oplus, \otimes, \top, \bot)$ such that $(L, \oplus, \otimes)$ is a lattice and $\top$ ($\bot$) is the identity element for the meet (join) operation. Both algebraic structures have non-overlapping properties: in a commutative semiring $\cdot$ distributes over $+$ and in a lattice absorption laws and idempotent laws hold (see Section 2.2). So the special case of a bounded, distributive lattice is a commutative, idempotent semiring under join and meet. It has been shown in [Tan10; GKT07] that five other provenance models from database literature are subsumed by their theory based on commutative semirings.

In this section, we introduce three different black-box approaches for computing a boundary. The first approach uses an axiom pinpointing algorithm as black-box reasoner, whereas the second one modifies the Hitting-Set-Tree-based black-box approach to axiom pinpointing [Kal+07; Sun08]. The third uses binary search and can only be applied if the labeling lattice is a linear order. It can be seen as a generalization of the black-box approach to reasoning in possibilistic Description Logics described in [QPJ07]. Our experimental results in Section 6.2.2 show that our algorithms perform well in practice.

4.2.1 Applicable Ontology Languages

To stay as general as possible, we do not fix a specific ontology language. We just assume that *ontologies* are finite sets of *axioms* such that every subset of an ontology is again an ontology. If O' is a subset of the ontology O, then O' is called a *sub-ontology* of O. An ontology language specifies which sets of axioms are admitted as ontologies. Consider, for instance, a Description Logic $\mathcal{L}$ (e.g., the DL $\mathcal{SHOIN}(\mathbf{D})$ underlying OWL DL). Then, an ontology is a finite set of general concept inclusion axioms (GCIs) of the form $C \sqsubseteq D$, with C, D $\mathcal{L}$-concept descriptions and assertion axioms of the form $C(a)$, with C an $\mathcal{L}$-concept description and a an individual name. Examples of consequences are subsumption relations $A \sqsubseteq B$ for concept names A, B. For a fixed ontology language, a *monotone consequence relation* $\models$ is a binary relation between ontologies O of this language and *consequences* c such that, for every ontology O, we have that $O' \subseteq O$ and $O' \models c$ imply $O \models c$.

4.2.2 Sub-Ontologies and Labels

We consider a lattice $(L, \leqslant)$ and respectively denote by $\bigoplus_{\ell \in S} \ell$ and $\bigotimes_{\ell \in S} \ell$ the *join* (least upper bound) and *meet* (greatest lower bound) of the finite set $S \subseteq L$. A *labeled ontology with labeling lattice* $(L, \leqslant)$ is an ontology O together with a labeling function lab that assigns a *label* $\mathsf{lab}(a) \in L$ to every element a of O.[3] We denote with L_{lab} the set of all labels occurring in the labeled ontology O, i.e., $L_{\mathsf{lab}} := \{\mathsf{lab}(a) \mid a \in O\}$. Every element of the labeling lattice $\ell \in L$ defines a sub-ontology $O_{\geqslant \ell}$ that contains the axioms of O that are labeled with elements greater than or equal to ℓ:

$$O_{\geqslant \ell} := \{a \in O \mid \mathsf{lab}(a) \geqslant \ell\}.$$

[3] An example of a labeled ontology is given in Example 1.

Conversely, every sub-ontology $S \subseteq O$ defines an element $\lambda_S \in L$, called the *label* of S: $\lambda_S := \bigotimes_{a \in S} \mathsf{lab}(a)$. The following lemma states some simple relationships between these two notions.

Lemma 1. *For all $\ell \in L$, $S \subseteq O$, it holds that $\ell \leqslant \lambda_{O_{\geqslant \ell}}$, $S \subseteq O_{\geqslant \lambda_S}$ and $O_{\geqslant \ell} = O_{\geqslant \lambda_{O_{\geqslant \ell}}}$.*

Proof. For the first statement, notice that, by definition, $\ell \leqslant \mathsf{lab}(a)$ for all $a \in O_{\geqslant \ell}$. Hence, $\ell \leqslant \bigotimes_{a \in O_{\geqslant \ell}} \mathsf{lab}(a) = \lambda_{O_{\geqslant \ell}}$. Regarding the second claim, if $a \in S$, then $\lambda_S = \bigotimes_{s \in S} \mathsf{lab}(s) \leqslant \mathsf{lab}(a)$, which implies that $a \in O_{\geqslant \lambda_S}$. Now, consider the last claim. First, as $\ell \leqslant \lambda_{O_{\geqslant \ell}}$, it holds trivially that $O_{\geqslant \lambda_{O_{\geqslant \ell}}} \subseteq O_{\geqslant \ell}$. From the second claim it also follows that $O_{\geqslant \ell} \subseteq O_{\geqslant \lambda_{O_{\geqslant \ell}}}$. □

4.2.3 Restrictions to User Labels

Notice that, if a consequence c follows from $O_{\geqslant \ell}$ for some $\ell \in L$, it must also follow from $O_{\geqslant \ell'}$ for every $\ell' \leqslant \ell$, since then $O_{\geqslant \ell} \subseteq O_{\geqslant \ell'}$. A maximal element of L that still entails the consequence will be called a margin for this consequence.

Definition 26 (Margin)**.** Let c be a consequence that follows from the ontology O. The label $\mu \in L$ is called a (O, c)-*margin* if $O_{\geqslant \mu} \models c$, and for every ℓ with $\mu < \ell$ we have $O_{\geqslant \ell} \not\models c$.

If O and c are clear from the context, we usually ignore the prefix (O, c) and call μ simply a *margin*. The following lemma shows three basic properties of the set of margins that will be useful throughout this section.

Lemma 2. *Let c be a consequence that follows from the ontology O. We have:*

1. *If μ is a margin, then $\mu = \lambda_{O_{\geqslant \mu}}$;*
2. *if $O_{\geqslant \ell} \models c$, then there is a margin μ such that $\ell \leqslant \mu$;*
3. *there are at most $2^{|O|}$ margins for c.*

Proof. To show 1, let $\mu \in L$. Lemma 1 yields $\mu \leqslant \lambda_{O_{\geqslant \mu}}$ and $O_{\geqslant \mu} = O_{\geqslant \lambda_{O_{\geqslant \mu}}}$, and thus $O_{\geqslant \lambda_{O_{\geqslant \mu}}} \models c$. If $\mu < \lambda_{O_{\geqslant \mu}}$, then this $\lambda_{O_{\geqslant \mu}}$ contradicts our assumption that μ is a margin; hence $\mu = \lambda_{O_{\geqslant \mu}}$. Point 3 is a trivial consequence of 1: since every margin has to be of the form λ_S for some $S \subseteq O$, there are at most as many margins as there are subsets of O.

For the remaining point, let $\ell \in L$ be such that $O_{\geqslant \ell} \models c$. Let $m := \lambda_{O_{\geqslant \ell}}$. From Lemma 1, it follows that $\ell \leqslant m$ and $O_{\geqslant m} = O_{\geqslant \ell}$, and hence $O_{\geqslant m} \models c$. If m is a margin, then the result holds; suppose to the contrary that m is not a margin. Then, there must exist an $\ell_1, m < \ell_1$, such that $O_{\geqslant \ell_1} \models c$. As $m = \lambda_{O_{\geqslant m}}$, there must exist an axiom $a \in O$ such that $m \leqslant \mathsf{lab}(a)$, but $\ell_1 \not\leqslant \mathsf{lab}(a)$. In fact, if $m \leqslant \mathsf{lab}(a) \implies \ell_1 \leqslant \mathsf{lab}(a)$ would hold for all $a \in O$, then $m = \lambda_{O_{\geqslant \ell}} = \lambda_{O_{\geqslant m}} = \bigotimes_{\mathsf{lab}(a) \geqslant m} \mathsf{lab}(a) \geqslant \ell_1$, contradicting our choice of ℓ_1. The existence of such an axiom a implies that $O_{\geqslant \ell_1} \subset O_{\geqslant m}$. Let $m_1 := \lambda_{O_{\geqslant \ell_1}}$; then $m < \ell_1 \leqslant m_1$. If m_1 is not a margin, then we can repeat the same process to obtain a new m_2 with $m < m_1 < m_2$ and $O_{\geqslant m} \supset O_{\geqslant m_1} \supset O_{\geqslant m_2}$, and so on. As O is finite, there exists a finite k where this process stops, and hence m_k is a margin. □

If we know that μ is a margin for the consequence c, then we know whether c follows from $O_{\geqslant \ell}$ for all $\ell \in L$ that are comparable with μ: if $\ell \leqslant \mu$, then c follows from $O_{\geqslant \ell}$, and if $\ell > \mu$, then c does not follow from $O_{\geqslant \ell}$. However, the fact that μ is a margin gives us no information regarding elements that are incomparable with μ. In order to obtain a full picture of when the consequence c follows from $O_{\geqslant \ell}$ for an arbitrary element of l, we can try to strengthen the notion of margin to that of an element ν of L that accurately divides the lattice into those elements whose associated sub-ontology entails c and those for which this is not the case, i.e., ν should satisfy the following: for every $\ell \in L$, $O_{\geqslant \ell} \models c$ iff $\ell \leqslant \nu$. Unfortunately, such an element need not always exist, as demonstrated by the following example.

Example 13. Consider the distributive lattice $(S_4, \leqslant_4)$ having the four elements $S_4 = \{0, \ell_1, \ell_2, 1\}$, where 0 and 1 are the least and greatest elements, respectively, and ℓ_1, ℓ_2 are incomparable w.r.t. $\leqslant_4$. Let O be the set formed by the axioms a_1 and a_2, which are labeled by ℓ_1 and ℓ_2, respectively, and let c be a consequence such that, for every $S \subseteq O$, we have $S \models c$ iff $|S| \geqslant 1$. It is easy to see that there is no element $\nu \in S_4$ that satisfies the condition described above. Indeed, if we choose $\nu = 0$ or $\nu = \ell_1$, then ℓ_2 violates the condition, as $\ell_2 \not\leqslant \nu$, but $O_{\geqslant \ell_2} = \{a_2\} \models c$. Accordingly, if we choose $\nu = \ell_2$, then ℓ_1 violates the condition. Finally, if $\nu = 1$ is chosen, then 1 itself violates the condition: $1 \leqslant \nu$, but $O_{\geqslant 1} = \emptyset \not\models c$.

It is nonetheless possible to find an element that satisfies a restricted version of the condition, where we do not impose that the property must hold for every element of the labeling lattice, but only for those elements that are *join prime* relative to the labels of the axioms in the ontology.

Definition 27 (Join prime)**.** Let $(L, \leqslant)$ be a lattice. Given a finite set $K \subseteq L$, let $K_{\otimes} := \{\bigotimes_{\ell \in M} \ell \mid M \subseteq K\}$ denote the closure of K under the meet operator. An element $\ell \in L$ is called *join prime relative to* K if, for every $K' \subseteq K_{\otimes}$, $\ell \leqslant \bigoplus_{k \in K'} k$ implies that there is an $k_0 \in K'$ such that $\ell \leqslant k_0$.

In Example 13, all lattice elements with the exception of 1 are join prime relative to $\{\ell_1, \ell_2\}$. In Example 1 all the elements of the labeling lattice except ℓ_1 and ℓ_4 are join prime relative to the set of labels.

Definition 28 (Boundary)**.** Let O be an ontology and c a consequence. An element $\nu \in L$ is called a (O, c)-*boundary* if for every element $\ell \in L$ that is join prime relative to L_{lab} it holds that $\ell \leqslant \nu$ iff $O_{\geqslant \ell} \models c$.

As with margins, if O and c are clear from the context, we will simply call such a ν a *boundary*. When it is clear that the computed boundary and no assigned label is meant, we also often call it *consequence label*. In Example 13, the element 1 is a boundary. Indeed, every join prime element ℓ relative to $\{\ell_1, \ell_2\}$ (i.e., every element of L except for 1) is such that $\ell < 1$ and $O_{\geqslant \ell} \models c$. From a practical point of view, our definition of a boundary has the following implication: we must enforce that user labels are always join prime relative to the set L_{lab} of all labels occurring in the ontology. The set of all user labels is denoted as U. In Example 1 all the elements of the labeling lattice except ℓ_1 and ℓ_4 are join prime relative to L_{lab} and for this reason $\ell_0, \ell_2, \ell_3, \ell_5$

are valid user labels and represent user roles as illustrated. Given a user label ℓ_u, we will say that the user *sees* a consequence c if $\ell_u \leqslant \nu$ for some boundary ν.

4.2.4 Computing a Consequence's Label

In this section, we describe three black-box approaches for computing a boundary. The first two approaches are based on Lemma 3 below, and the third one, a modification of binary search, can be used if the labeling lattice is a linear order.

Lemma 3. *Let $\mu_1, \ldots, \mu_n$ be all (O, c)-margins. Then $\bigoplus_{i=1}^{n} \mu_i$ is a boundary.*

Proof. Let $\ell \in L$ be join prime relative to L_{lab}. We need to show that $\ell \leqslant \bigoplus_{i=1}^{n} \mu_i$ iff $O_{\geqslant \ell} \models c$. Assume first that $O_{\geqslant \ell} \models c$. Then, from 2 of Lemma 2, it follows that there is a margin μ_j such that $\ell \leqslant \mu_j$, and thus $\ell \leqslant \bigoplus_{i=1}^{n} \mu_i$.

Conversely, let $\ell \leqslant \bigoplus_{i=1}^{n} \mu_i$. From 1 of Lemma 2, it follows that for every $i, 1 \leqslant i \leqslant n, \mu_i \in (L_{\mathsf{lab}})_{\otimes}$. As ℓ is join prime relative to L_{lab}, it then holds that there is a j such that $\ell \leqslant \mu_j$ and hence, by the definition of a margin and the monotonicity of the consequence relation, $O_{\geqslant \ell} \models c$. □

By Lemma 2, a consequence always has finitely many margins, and thus Lemma 3 shows that a boundary always exists. Note, however, that a consequence may have boundaries different from the one of Lemma 3. To identify the particular boundary of Lemma 3, we will call it the *margin-based boundary*.

To address consequence labels in a convenient way, we define the function lbl.

Definition 29 (Consequence Labeling Function)**.** Let O be a labeled ontology, $(L, \leqslant)$ a labeling lattice, $\mathsf{lab} : O \to L$ a labeling function. The *consequence labeling function* $\mathsf{lbl} : \{c \mid O \models c\} \to L$ assigns labels to consequences and is defined as $\mathsf{lbl}(c) =$ margin-based boundary of c.

Using Full Axiom Pinpointing

From Lemma 3 we know that the set of all margins yields sufficient information for computing a boundary. The question is now how to compute this set. In this subsection, we show that all margins (and thus the margin-based boundary) can be computed through *axiom pinpointing*. Axiom-pinpointing and the notion of *MinA* and *diagnosis* have been introduced in Section 2.1. The following lemma shows that every margin can be obtained from some MinA.

Lemma 4. *For every margin μ for c there is a MinA S such that $\mu = \lambda_S$.*

Proof. If μ is a margin, then $O_{\geqslant \mu} \models c$ by definition. Thus, there exists a MinA $S \subseteq O_{\geqslant \mu}$. Since $\mu \leqslant \mathsf{lab}(a)$ for every $a \in O_{\geqslant \mu}$, this in particular holds also for every axiom in S, and hence $\mu \leqslant \lambda_S$. Additionally, as $S \subseteq O_{\geqslant \lambda_S}$, we have $O_{\geqslant \lambda_S} \models c$. This implies $\mu = \lambda_S$ since otherwise $\mu < \lambda_S$, and then μ would not be a margin. □

Notice that this lemma does not imply that the label of any MinA S corresponds to a margin. However, as the consequence follows from every MinA, point 2 of Lemma 2 shows that $\lambda_S \leqslant \mu$ for some margin μ. The following theorem is an immediate consequence of this fact together with Lemma 3 and Lemma 4.

Lemma 5. *If $S_1, \ldots, S_n$ are all MinAs for O and c, then $\bigoplus_{i=1}^{n}(\bigotimes_{a \in S_i} \mathsf{lab}(a))$ is the margin-based boundary for c.*

This lemma, together with the Definition 23 on page 36, trivially implies that for a given consequence, assigning its margin-based boundary as label ℓ_c is an inference-proof label assignment, because for any MinA $\{a_1, \ldots, a_n\}$ where $\mathsf{lab}(a_i) = \ell_i$ it holds that $\ell_c \geqslant \ell_1 \otimes \ldots \otimes \ell_n$.

A dual result, which relates the boundary with the set of diagnoses, also exists. The proof follows easily from the definitions given in this section.

Lemma 6. *If $S_1, \ldots, S_n$ are all diagnoses for O and c, then $\bigotimes_{i=1}^{n}(\bigoplus_{a \in S_i} \mathsf{lab}(a))$ is a boundary for c.*

Example 14. We continue Example 1 where each axiom a_i is labeled with $\mathsf{lab}(a_i) = \ell_i$. We are interested in the access restriction to the inferred instance relation c : *ServiceWithComingPriceIncrease*(*ecoCalculatorV1*), which has four MinAs, namely $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}, \{a_1, a_3, a_5\}$, and 3 diagnoses $\{a_1\}, \{a_2, a_3\}, \{a_4, a_5\}$. Using Lemma 5, we can compute the boundary as $(\ell_1 \otimes \ell_2 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_2 \otimes \ell_5) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_5) = \ell_3 \oplus \ell_0 \oplus \ell_3 \oplus \ell_0 = \ell_3$. Using Lemma 6, we can compute the boundary as $\ell_1 \otimes (\ell_2 \oplus \ell_3) \otimes (\ell_4 \oplus \ell_5) = \ell_1 \otimes \ell_2 \otimes \ell_4 = \ell_3$. So c is visible to users with labels ℓ_0 or ℓ_3.

Thus, to compute a boundary, it is sufficient to compute all MinAs. Several methods exist for computing the set of all MinAs, either directly [SC03; Kal+07; BS08] or through a so-called pinpointing formula [BPS07; BP08; BP10b], which is a monotone Boolean formula encoding all the MinAs. The main advantage of using the pinpointing-based approach for computing a boundary is that one can simply use existing implementations for computing all MinAs, such as the ones offered by the ontology editor Protégé 4[4] and the CEL system.[5]

Label-Optimized Axiom Pinpointing

From Lemma 4 we know that every margin is of the form λ_S for some MinA S. In the previous subsection we have used this fact to compute a boundary by first obtaining the MinAs and then computing their labels. This process can be optimized if we directly compute the labels of the MinAs, without necessarily computing the actual MinAs. Additionally, not all the labels of MinAs are necessary, but only the maximal ones. We present here a black-box algorithm that uses the labels of the axioms to find the boundary in an optimized way. Our algorithm is a variant of the Hitting-Set-Tree-based [Rei87] method (HST approach) for axiom pinpointing [Kal+07; Sun08]. First, we briefly describe the HST approach for computing all MinAs, which will serve as a starting point for our modified version.

The HST-based method for axiom pinpointing computes one MinA at a time while building a tree that expresses the distinct possibilities to be explored in the search of further MinAs. It first computes an arbitrary MinA S_0 for O, which is used to label

[4] http://protege.stanford.edu/
[5] http://code.google.com/p/cel/

the root of the tree. Then, for every axiom a in S_0, a successor node is created. If $O \setminus \{a\}$ does not entail the consequence, then this node is a dead end. Otherwise, $O \setminus \{a\}$ still entails the consequence. In this case, a MinA S_1 for $O \setminus \{a\}$ is computed and used to label the node. The MinA S_1 for $O \setminus \{a\}$ obtained this way is also a MinA of O, and it is guaranteed to be distinct from S_0 since $a \notin S_1$. Then, for each axiom s in S_1, a new successor is created, and treated in the same way as the successors of the root node, i.e., it is checked whether $O \setminus \{a, s\}$ still has the consequence, etc. This process obviously terminates since O is a finite set of axioms, and the end result is a tree, where each node that is not a dead end is labeled with a MinA, and every existing MinA appears as the label of at least one node of the tree (see [Kal+07; Sun08]).

An important ingredient of the HST algorithm is a procedure that computes a single MinA from an ontology. Such a procedure can, e.g., be obtained by going through the axioms of the ontology in an arbitrary order, and removing redundant axioms, i.e., ones such that the ontology obtained by removing this axiom from the current sub-ontology still entails the consequence. This procedure would also be an option to manually identify a MinA with a reasoner at hand which decides whether a consequence follows from a given set of axioms. See [BPS07] for a description of this and of a more sophisticated logarithmic procedure.

Although not stated explicitly in the axiom pinpointing literature, it is clear that the same HST algorithm can be used for computing all diagnoses. The only variant necessary is to have a subroutine capable of computing one such diagnosis, which can be obtained by dualizing the algorithm computing one MinA.

As said before, in our modified HST algorithm, we are now not interested in actually computing a MinA, but only its label. This allows us to remove all axioms having a "redundant" label rather than a single axiom. Algorithm 4.1 describes a black-box method for computing λ_S for some MinA S that is based on this idea. In fact, the algorithm computes a *minimal label set (MinLab)* of a MinA S, a notion that will also be useful when describing our variant of the HST algorithm.

Definition 30 (Minimal label set)**.** Let S be a MinA for c. A set $K \subseteq \{\mathsf{lab}(a) \mid a \in S\}$ is called a *minimal label set* of S if distinct elements of K are incomparable and $\lambda_S = \bigotimes_{\ell \in K} \ell$.

Algorithm 4.1 removes all the labels that do not contribute to a minimal label set. If O is an ontology and $\ell \in L$, then the expression $O - \ell$ appearing at Line 7 denotes the sub-ontology $O - \ell := \{a \in O \mid \mathsf{lab}(a) \neq \ell\}$. If, after removing all the axioms labeled with k, the consequence still follows, then there is a MinA none of whose axioms is labeled with k. In particular, this MinA has a minimal label set not containing k; thus all the axioms labeled with k can be removed in our search for a minimal label set. If the axioms labeled with k cannot be removed, then all MinAs of the current sub-ontology need an axiom labeled with k, and hence k is stored in the set M_L. This set is used to avoid useless consequence tests: if a label is greater than or equal to $\bigotimes_{\ell \in M_L} \ell$, then the presence or absence of axioms with this label will not influence the final result, which will be given by the infimum of M_L; hence, there is no need to apply the (possibly complex) decision procedure for the consequence relation.

Algorithm 4.1 Compute a minimal label set of one MinA

Procedure min-lab(O, c)
Input: O: ontology; c: consequence
Output: $M_L \subseteq L$: minimal label set for a MinA

```
1: if O ⊭ c then
2:    return no MinA
3: S := O
4: M_L := ∅
5: for every k ∈ L_lab do
6:    if ⨂_{l∈M_L} l ⋠ k then
7:       if S − k ⊨ c then
8:          S := S − k
9:       else
10:         M_L := (M_L \ {l | k < l}) ∪ {k}
11: return M_L
```

Theorem 1. *Let O and c be such that $O \models c$. There is a MinA S_0 for c such that Algorithm 4.1 outputs a minimal label set of S_0.*

Proof. As $O \models c$, the algorithm will enter the **for** loop. This loop keeps the following two invariants: (i) $S \models c$ and (ii) for every $\ell \in M_L, S - \ell \not\models c$. The invariant (i) is ensured by the condition in Line 7 that must be satisfied before S is modified. Otherwise, that is, if $S - \ell \not\models c$, then ℓ is added to M_L (Line 10) which, together with the fact that S is always modified to a smaller set (Line 8), ensures (ii). Hence, when the loop finishes, the sets S and M_L satisfy both invariants. As $S \models c$, there is a MinA $S_0 \subseteq S$ for c. For each $\ell \in M_L$, there must be an axiom $a \in S_0$ such that $\mathsf{lab}(a) = \ell$, otherwise, $S_0 \subseteq S - \ell$ and hence $S - \ell \models c$, which contradicts invariant (ii); thus, $M_L \subseteq \{\mathsf{lab}(a) \mid a \in S_0\}$ and in particular $\lambda_{S_0} \leqslant \bigotimes_{\ell \in M_L} \ell$.

It remains to show that the inequality in the other direction holds as well. Consider now $k \in \{\mathsf{lab}(a) \mid a \in S\}$ and let M_L^k be the value of M_L when the **for** loop was entered with value k. We have that $\bigotimes_{\ell \in M_L} \ell \leqslant \bigotimes_{\ell \in M_L^k} \ell$. If $\bigotimes_{\ell \in M_L} \ell \not\leqslant k$, then also $\bigotimes_{\ell \in M_L^k} \ell \not\leqslant k$, and thus it fulfills the test in Line 6, and continues to Line 7. If that test is satisfied, then all the axioms with label k are removed from S, contradicting the assumption that $k = \mathsf{lab}(a)$ for some $a \in S$. Otherwise, k is added to M_L, which contradicts the assumption that $\bigotimes_{\ell \in M_L} \ell \not\leqslant k$. Thus, for every axiom a in S, $\bigotimes_{\ell \in M_L} \ell \leqslant \mathsf{lab}(a)$; hence $\bigotimes_{\ell \in M_L} \ell \leqslant \lambda_S \leqslant \lambda_{S_0}$. □

Once the label of a MinA has been found, we can compute new MinA labels by a successive deletion of axioms from the ontology using the HST approach. Suppose that we have computed a minimal label set $\mathcal{M}_0$, and that $\ell \in \mathcal{M}_0$. If we remove all the axioms in the ontology labeled with ℓ, and compute a new minimal label set $\mathcal{M}_1$ of a MinA of this sub-ontology, then $\mathcal{M}_1$ does not contain ℓ, and thus $\mathcal{M}_0 \neq \mathcal{M}_1$. By iterating this procedure, we could compute all minimal label sets, and hence the labels of all MinAs. However, since our goal is to compute the supremum of these labels, the algorithm can be optimized by avoiding the computation of MinAs whose labels will

have no impact on the final result. Based on this we can actually do better than just removing the axioms with label ℓ: instead, all axioms with labels $\leqslant \ell$ can be removed. For an element $\ell \in L$ and an ontology O, $O_{\not\leqslant \ell}$ denotes the sub-ontology obtained from O by removing all axioms whose labels are $\leqslant \ell$. Now, assume that we have computed the minimal label set $\mathcal{M}_0$, and that $\mathcal{M}_1 \neq \mathcal{M}_0$ is the minimal label set of the MinA S_1. For all $\ell \in \mathcal{M}_0$, if S_1 is not contained in $O_{\not\leqslant \ell}$, then S_1 contains an axiom with label $\leqslant \ell$. Consequently, $\bigotimes_{m \in \mathcal{M}_1} m = \lambda_{S_1} \leqslant \bigotimes_{m \in \mathcal{M}_0} m$, and thus $\mathcal{M}_1$ need not be computed. Algorithm 4.2 describes our method for computing the boundary using a variant of the HST algorithm that is based on this idea.

Algorithm 4.2 Compute a boundary by a HST algorithm

Procedure HST-boundary(O, c)
Input: O: ontology; c: consequence
Output: boundary ν for c

1: **Global** : $\mathbf{C}, \mathbf{H} := \emptyset; \nu$
2: $\mathcal{M} :=$ min-lab(O, c)
3: $\mathbf{C} := \{\mathcal{M}\}$
4: $\nu := \bigotimes_{\ell \in \mathcal{M}} \ell$
5: **for** each label $\ell \in \mathcal{M}$ **do**
6: expand-HST$(O_{\not\leqslant \ell}, c, \{\ell\})$
7: **return** ν

Procedure expand-HST(O, c, H)
Input: O: ontology; c: consequence; H: list of lattice elements
Side effects: modifications to $\mathbf{C}$, $\mathbf{H}$ and ν

1: **if** there exists some $H' \in \mathbf{H}$ such that $\{h \in H' \mid h \not\leqslant \nu\} \subseteq H$ **or**
H' contains a prefix-path P with $\{h \in P \mid h \not\leqslant \nu\} = H$ **then**
2: **return** (early path termination $\otimes$)
3: **if** there exists some $\mathcal{M} \in \mathbf{C}$ such that for all $\ell \in \mathcal{M}, h \in H$, $\ell \not\leqslant h$ and $\ell \not\leqslant \nu$ **then**
4: $\mathcal{M}' := \mathcal{M}$ (MinLab reuse)
5: **else**
6: $\mathcal{M}' :=$ min-lab$(O_{\not\leqslant \nu}, c)$
7: **if** $O_{\not\leqslant \nu} \models c$ **then**
8: $\mathbf{C} := \mathbf{C} \cup \{\mathcal{M}'\}$
9: $\nu := \bigoplus\{\nu, \bigotimes_{\ell \in \mathcal{M}'} \ell\}$
10: **for** each label $\ell \in \mathcal{M}'$ **do**
11: expand-HST$(O_{\not\leqslant \ell}, c, H \cup \{\ell\})$
12: **else**
13: $\mathbf{H} := \mathbf{H} \cup \{H\}$ (normal termination $\odot$)

In the procedure HST-boundary, three global variables are declared: $\mathbf{C}$ and $\mathbf{H}$, initialized with $\emptyset$, and ν. The variable $\mathbf{C}$ stores all the minimal label sets computed so far, while each element of $\mathbf{H}$ is a set of labels such that, when all the axioms with a label less than or equal to any label from the set are removed from the ontology, the consequence does not follow anymore; the variable ν stores the supremum of the labels

of all the elements in $\mathbf{C}$ and ultimately corresponds to the boundary that the method computes. The algorithm starts by computing a first minimal label set $\mathcal{M}$, which is used to label the root of a tree. For each element of $\mathcal{M}$, a branch is created by calling the procedure expand-HST.

The procedure expand-HST implements the ideas of HST construction for pinpointing [Kal+07; Sun08] with additional optimizations that help reduce the search space as well as the number of calls to min-lab. First notice that each $\mathcal{M} \in \mathbf{C}$ is a minimal label set, and hence the infimum of its elements corresponds to the label of some MinA for c. Thus, ν is the supremum of the labels of a set of MinAs for c. If this is not yet the boundary, then there must exist another MinA S whose label is not less than or equal to ν. This in particular means that no element of S may have a label less than or equal to ν, as the label of S is the infimum of the labels of the axioms in it. When searching for this new MinA we can then exclude all axioms having a label $\leqslant \nu$, as done in Line 6 of expand-HST. Every time we expand a node, we extend the set H, which stores the labels that have been removed on the path in the tree to reach the current node. If we reach normal termination, it means that the consequence does not follow anymore from the reduced ontology. Thus, any H stored in $\mathbf{H}$ is such that, if all the axioms having a label less than or equal to an element in H are removed from O, then c does not follow anymore. Lines 1 to 4 of expand-HST are used to reduce the number of calls to the subroutine min-lab and the total search space. We describe them now in more detail.

The first optimization, *early path termination*, prunes the tree once we know that no new information can be obtained from further expansion. There are two conditions that trigger this optimization. The first one tries to decide whether $O_{\not\leqslant \nu} \models c$ without executing the decision procedure. As said before, we know that for each $H' \in \mathbf{H}$, if all labels less than or equal to any in H' are removed, then the consequence does not follow. Hence, if the current list of removal labels H contains a set $H' \in \mathbf{H}$ we know that enough labels have been removed to make sure that the consequence does not follow. It is actually enough to test whether $\{h \in H' \mid h \not\leqslant \nu\} \subseteq H$ since the consequence test we need to perform is whether $O_{\not\leqslant \nu} \models c$. The second condition for early path termination asks for a prefix-path P of H' such that $P = H$. If we consider H' as a list of elements, then a prefix-path is obtained by removing a final portion of this list. The idea is that, if at some point we have noticed that we have removed the same axioms as in a previous portion of the search, we know that all possibilities that arise from that search have already been tested before, and hence it is unnecessary to repeat the work. Hence we can prune the tree here. As an example, consider a subtree reachable from the root by going along the edges ℓ_1, ℓ_2 which has been expanded completely. Then all Hitting Sets of its leaf nodes share the common prefix-path $P = \{\ell_1, \ell_2\}$. Now suppose the tree is expanded by expand-HST(O, c, H) with $H = \{\ell_2, \ell_1\}$. The expansion stops with early termination since $P = H$.

The second optimization avoids a call to min-lab by *reusing* a previously computed minimal label set. Notice that our only requirement on min-lab is that it produces a minimal label set. Hence, any minimal label set for the ontology obtained after removing all labels less than or equal to any $h \in H$ or to ν would work. The MinLab reuse optimization checks whether there is such a previously computed minimal label

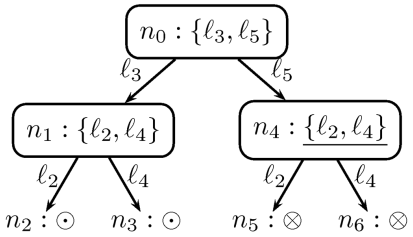

Figure 4.1: An expansion of the HST method

set. If this is the case, it uses this set instead of computing a new one by calling min-lab. If we would leave out the prefix-path condition for early termination, still the MinLab reuse condition holds. That means leaving out the prefix-path condition leads to no more min-lab calls but leads to copying in the tree without obtaining new information.

Before showing that the algorithm is correct, we illustrate how it works by a small example.

Example 15. We continue Example 14 with the already introduced consequence c : *ServiceWithComingPriceIncrease*(*ecoCalculatorV1*). Figure 4.1 shows a possible run of the HST-boundary algorithm. The algorithm first calls the routine min-lab(O, c). Consider that the **for** loop of min-lab is executed using the labels in the order $\ell_1, \ell_2, \ell_4, \ell_3, \ell_5$ since Line 5 requires no specific order. Thus, we try first to remove a_1 labeled with ℓ_1. We see that $O - \ell_1 \not\models c$; hence a_1 is not removed from O, and M_L is updated to $M_L = \{\ell_1\}$. We then see that $O - \ell_2 \models c$, and thus a_2 is removed from O. Again, $O - \ell_4 \models c$, so a_4 is removed from O. At this point, $O = \{a_1, a_3, a_5\}$. We test then whether $O - \ell_3 \models c$ and receive a negative answer; thus, ℓ_3 is added to M_L; additionally, since $\ell_3 < \ell_1$, the latter is removed from M_L. Finally, $O - \ell_5 \not\models c$, and so we obtain $M_L = \{\ell_3, \ell_5\}$ as an output of min-lab.

The minimal label set $\{\ell_3, \ell_5\}$, is used as the root node n_0, setting the value of $\nu = \ell_3 \otimes \ell_5 = \ell_0$. We then create the first branch on the left by removing all the axioms with a label $\leqslant \ell_3$, which is only a_3, and computing a new minimal label set. Assume, for the sake of the example, that min-lab returns the minimal label set $\{\ell_2, \ell_4\}$, and ν is accordingly changed to ℓ_3. When we expand the tree from this node, by removing all the axioms below ℓ_2 (left branch) or ℓ_4 (right branch), the instance relation c does not follow any more, and hence we have a normal termination, adding the sets $\{\ell_3, \ell_2\}$ and $\{\ell_3, \ell_4\}$ to $\mathbf{H}$. We then create the second branch from the root, by removing the elements below ℓ_5. We see that the previously computed minimal label set of node n_1 works also as a minimal label set in this case, and hence it can be reused (MinLab reuse), represented as an underlined set. The algorithm continues now by calling expand-HST$(O_{\not\leqslant \ell_2}, c, \{\ell_5, \ell_2\})$. At this point, we detect that there is $H' = \{\ell_3, \ell_2\}$ satisfying the first condition of early path termination (recall that $\nu = \ell_3$), and hence the expansion of that branch at that point. Analogously, we obtain an early path termination on the second expansion branch of the node n_4. The algorithm then outputs $\nu = \ell_3$, which can be easily verified to be a boundary.

Theorem 2. *Let O and c be such that $O \models c$. Then Algorithm 4.2 computes the margin-based boundary of c.*

Proof. Let η be the margin-based boundary which, by Lemma 3, must exist. Notice first that the procedure expand-HST keeps as invariant that $\nu \leqslant \eta$ as whenever ν is modified, it is only to join it with the infimum of a minimal label set (Line 9), which by definition is the label of a MinA and, by Lemma 5, is $\leqslant \eta$. Thus, when the algorithm terminates, we have that $\nu \leqslant \eta$. Assume now that $\nu \neq \eta$. Then, there must exist a MinA S such that $\lambda_S \not\leqslant \nu$; in particular, this implies that none of the axioms in S has a label $\leqslant \nu$ and thus $S \subseteq O_{\not\leqslant \nu}$. Let $\mathcal{M}_0$ be the minimal label set obtained in Line 2 of HST-boundary. There must then be a $h_0 \in \mathcal{M}_0$ such that $S \subseteq O_{\not\leqslant h_0}$; otherwise, $\lambda_S \leqslant \bigotimes_{\ell \in \mathcal{M}_0} \ell \leqslant \nu$. There will then be a call to the process expand-HST with parameters $O_{\not\leqslant h_0}, c$, and $\{h_0\}$. Suppose first that early path termination is not triggered. A minimal label set $\mathcal{M}_1$ is then obtained, either by MinLab reuse (Line 4) or by a call to min-lab (Line 6). As before, there is a $h_1 \in \mathcal{M}_1$ with $S \subseteq (O_{\not\leqslant h_0})_{\not\leqslant h_1}$. Additionally, since $O_{\not\leqslant h_0}$ does not contain any axiom labeled with h_0, we know $h_0 \notin \mathcal{M}_1$. While iterating this algorithm, we can find a sequence of minimal label sets $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ and labels $h_0, h_1, \ldots, h_n$ such that (i) $h_i \in \mathcal{M}_i$, (ii) $S \subseteq O_{\not\leqslant h_i}$, and (iii) $h_i \notin \mathcal{M}_j$ for all i, j, $1 \leqslant i < j \leqslant n$. In particular, this means that the $\mathcal{M}_i$s are all different, and since there are only finitely many minimal label sets, this process must terminate. Let $\mathcal{M}_n$ be the last set found this way. Then, when expand-HST is called with $\mathcal{R} := (((O_{\not\leqslant h_0})_{\not\leqslant h_1})_{\ldots})_{\not\leqslant h_n}, c$ and $H = \{h_1, \ldots, h_n\}$, no new minimal label set is found. Suppose first that this is due to a normal termination. Then, $\mathcal{R}_{\not\leqslant \nu} \not\models c$. But that contradicts the fact that S is a MinA for c since $S \subseteq \mathcal{R}_{\not\leqslant \nu}$. Hence, it must have finished by early termination.

There are two possible causes for early termination. Suppose first that there is a $H' \in \mathbf{H}$ such that $\{h \in H' \mid h \not\leqslant \nu\} \subseteq H$. Then it is also the case that, for every $h \in H'$, $S \subseteq O_{\not\leqslant h}$: if $h \in H$, then $\mathcal{R} \subseteq O_{\not\leqslant h}$; otherwise, $h \leqslant \nu$ and hence $O_{\not\leqslant \nu} \subseteq O_{\not\leqslant h}$. Let $\mathcal{R}' := \{a \in O \mid \text{there is no } h \in H' \text{ with } \mathsf{lab}(a) \leqslant h\}$. As $H' \in \mathbf{H}$, it was added after a normal termination; thus, c does not follow from $\mathcal{R}'_{\not\leqslant \nu}$. As $S \subseteq \mathcal{R}_{\not\leqslant \nu}$, we obtain once again a contradiction.

The second cause for early path termination is the existence of a prefix-path P with $\{h \in P \mid h \not\leqslant \nu\} = H$. This means that in a previously explored path we had concluded that $\mathcal{R}_{\not\leqslant \nu} \models c$, and a new minimal label set $\mathcal{M}_{n+1}$ was found. As in the beginning of this proof, we can then compute sets $\mathcal{M}_{n+1}, \ldots, \mathcal{M}_m$ and $h_{n+1}, \ldots, h_m$ $(n < m)$ such that $S \subseteq O_{\not\leqslant h_i}$ for all $i, 1 \leqslant i \leqslant m$ and the $\mathcal{M}_i$s are all different. Hence this process terminates. As before, the cause of termination cannot be normal termination, nor the first condition for early path termination. Thus, there must exist a new $H'' \in \mathbf{H}$ that fulfills the second condition for early termination. As $\mathbf{H}$ is a finite set, and each of its elements is itself a finite list, this process also terminates. When that final point is reached, there are no further causes of termination that do not lead to a contradiction, which means that our original assumption that $\nu \neq \eta$ cannot be true. Hence, ν is the margin-based boundary of c. □

Binary Search for Linear Ordering

In this subsection, we assume that the labeling lattice $(L, \leqslant)$ is a linear order, i.e., for any two elements ℓ_1, ℓ_2 of L we have $\ell_1 \leqslant \ell_2$ or $\ell_2 \leqslant \ell_1$.

Lemma 7. *Let O and c be such that $O \models c$. Then the unique boundary of c is the maximal element μ of L_{lab} with $O_{\geqslant \mu} \models c$.*

Proof. Let μ be the maximal element of L_{lab} with $O_{\geqslant \mu} \models c$. Such a maximal element exists since L_{lab} is finite. Obviously, $\ell \leqslant \mu$ implies $O_{\geqslant \ell} \supseteq O_{\geqslant \mu}$, and thus $O_{\geqslant \mu} \models c$ yields $O_{\geqslant \ell} \models c$. Conversely, assume that $O_{\geqslant \ell} \models c$. Then the fact that μ is maximal with this property together with the fact that $\leqslant$ is a linear order implies $\ell \leqslant \mu$. Thus, μ is a boundary. □

Algorithm 4.3 Compute a boundary by binary search.

Input: O: ontology; c: consequence
Output: ν: (O, c)-boundary

```
1: if O ⊭ c then
2:    return no boundary
3: ℓ := 0_lab; h := 1_lab
4: while l < h do
5:    set m, ℓ < m ⩽ h such that δ(ℓ, m) − δ(m, h) ⩽ 1.
6:    if O_{⩾m} ⊨ c then
7:       ℓ := m
8:    else
9:       h := pred(m)
10: return ν := ℓ
```

A direct way for computing the boundary in this restricted setting thus consists of testing, for every element in $\ell \in L_{\mathsf{lab}}$, in order (either increasing or decreasing) whether $O_{\geqslant \ell} \models c$ until the desired maximal element is found. This process requires in the worst case $n := |L_{\mathsf{lab}}|$ iterations. This can be improved using binary search, which requires a logarithmic number of steps measured in n. Algorithm 4.3 describes the binary search algorithm. In the description of the algorithm, the following abbreviations have been used: $\mathbf{0}_{\mathsf{lab}}$ and $\mathbf{1}_{\mathsf{lab}}$ represent the minimal and the maximal elements of L_{lab}, respectively; for $\ell_1 \leqslant \ell_2 \in L_{\mathsf{lab}}$, $\delta(\ell_1, \ell_2) := |\{\ell' \in L_{\mathsf{lab}} \mid \ell_1 < \ell' \leqslant \ell_2\}|$ is the *distance* function in L_{lab} and for a given $\ell \in L_{\mathsf{lab}}$, $\mathsf{pred}(\ell)$ is the maximal element $\ell' \in L_{\mathsf{lab}}$ such that $\ell' < \ell$.

The variables ℓ and h are used to keep track of the relevant search space. At every iteration of the **while** loop, the boundary is between ℓ and h. At the beginning these values are set to the minimum and maximum of L_{lab} and are later modified as follows: we first find the *middle* element m of the search space; i.e., an element whose distance to ℓ differs by at most one from the distance to h. We then test whether $O_{\geqslant m} \models c$. If that is the case, we know that the boundary must be larger or equal to m, and hence the lower bound ℓ is updated to the value of m. Otherwise, we know that the boundary is strictly smaller than m as m itself cannot be one; hence, the higher bound

h is updated to the maximal element of L_{lab} that is smaller than $m : \mathsf{pred}(m)$. This process terminates when the search space has been reduced to a single point, which must be the boundary.

An empirical evaluation of our algorithms from this section is provided in Section 6.2.1.

4.3 Repairing Access Restrictions to a Consequence

The last sections have shown that annotations are useful for representing access restrictions to the explicit axioms of an ontology. Intuitively, these access restrictions should have an effect also on the ontology's implicit consequences. Methods have been presented for assigning a label, representing its access level, to each consequence from a given ontology. However, a security administrator might not be satisfied with the access level obtained through these methods. In this case, one is interested in finding which axioms would need to get their access restrictions modified in order to receive the desired label for the consequence. In this section we look at this problem and present algorithms based on ontology repair for solving it with a variety of optimizations.

As introduced in the last sections, criteria for offering access to only a subset of an ontology can be: access rights, granularity, certainty, relevancy, trust, etc., without loss of generality we focus on access rights while the results remain applicable to all the other lattice-based applications. The criterion is formalized as labeling lattice $(L, \leqslant)$ with L the set of criteria levels and $\leqslant$ the order among those levels. An example might be a set of user roles and a permission inheritance relation among those user roles. As discussed in Section 4.2.3, a subset $U \subseteq L$ is allowed as user label. A labeled ontology is an ontology O where each contained axiom has a label from the set L. For a user labeled with $\ell \in L$, we denote as $O_{\geqslant \ell}$ the sub-ontology $O_{\geqslant \ell} := \{a \in O \mid \mathsf{lab}(a) \geqslant \ell\}$ visible for her. The sub-ontologies $O_{\leqslant \ell}, O_{=\ell}, O_{\neq \ell}, O_{\ngeqslant \ell}$, and $O_{\nleqslant \ell}$ are defined analogously. This notion is extended to sets of labels in the natural way, e.g. $O_{=K} := \{a \in O \mid \mathsf{lab}(a) = \ell$ for some $\ell \in K\}$. When dealing with labeled ontologies, the reasoning problem of interest consists on the computation of a boundary for a consequence c. Intuitively, the boundary divides the user labels ℓ of U according to whether $O_{\geqslant \ell}$ entails c or not.

Labeling an ontology's axioms to represent access restrictions falls into traditional access control. Computing a label for each consequence makes it possible to enforce access control also on implicit consequences which can be made explicit by a reasoner. However, the consequence label is determined by the axiom labeling. Determining it in the other direction, so that a given consequence label defines an axiom labeling, can be seen similar to the problem of reducing inference control to access control in databases [FJ02; BEL08]. Inference control assumes a set of defined secrets and checks at runtime on every response to a user's query whether this response together with the user's a priori knowledge and already delivered answers implies any secret. In contrast to that, access control is enforced by following a policy which regulates access on explicit data. For ontologies, we extended access control to implicit knowledge by computing a boundary for a consequence as described above. A security administrator can change the boundary only by changing the labels of axioms that entail the consequence. Now

given that she defines a consequence as a secret, she might be interested in support to find the smallest set of those axioms that must be secret in order to meet the goal.

Just as ontology development and maintenance is an error prone activity, so is the adequate labeling of axioms according to their access requirements. Indeed, several seemingly harmless axioms might possibly be combined to deduce knowledge that is considered private. For example, the architecture in Figure 1.7 contains a component to lift document access restrictions to axiom labels. If the document permissions have been set wrongly then reading all documents and especially all document ontologies could make it possible to infer consequences that are not intended to be available. On the other hand, an over-restrictive labeling of axioms may cause public knowledge to be inaccessible to some users.

If the knowledge engineer finds that the boundary for a given consequence differs from the desired one, then she would like to automatically receive suggestions on how to modify the labeling function and correct this error. In this section we present some methods in this direction. We assume that the knowledge engineer knows the exact boundary ℓ_g that the consequence c should receive, and propose a set S of axioms of minimal cardinality such that if all the axioms in S are relabeled to ℓ_g, then the boundary of c will be ℓ_g. We call S a *smallest change set*.

We show that the main ideas from axiom-pinpointing [SC03; Mey+06; Kal+05; BP10b; BP10a] can be exploited in the computation of a change set and present a Hitting Set Tree-based black-box approach that yields the desired set. The methods take advantage of our search of a set with minimum cardinality, as well as the axiom labeling to reduce the search space and hence also the execution time. Our experimental results in Section 6.2.2 show that our algorithms perform well in practice.

4.3.1 Modifying a Consequence's Label

Once the boundary for a consequence c has been computed, it is possible that the knowledge engineer or the security administrator considers this solution erroneous. For instance, the boundary may express that a given user u is able to deduce c, although this was not intended. Alternatively, the boundary may imply that c is a confidential consequence, only visible to a few, high-clearance users, while in reality c should be publicly available.

Example 16. We continue Example 1. The labeled ontology entails the consequence $c : \mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$, the computed boundary of c is ℓ_3 (see Example 14) which implies that only for ℓ_0 and ℓ_3, c is visible. That means the consequence c can only be seen by the development engineers and customer service employees (see Figure 1.3). It could be, however, that c is not expected to be accessible to customer service employees and development engineers, but rather to customer service employees and customers. In that case, we wish to modify the boundary of c to ℓ_5.

The problem we face is how to change the labeling function, so that the computed boundary corresponds to the desired label in the lattice. This problem can be formalized and approached in several different ways. In our approach, we fix a goal label

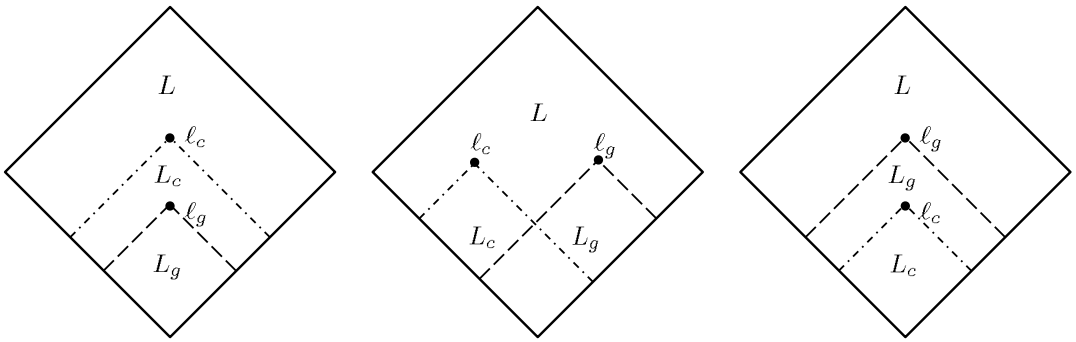

Figure 4.2: Hide consequence from some user roles (left), allow additional user roles to see consequence (right), and both at the same time (middle)

ℓ_g and try to modify the labeling of as few axioms as possible, so that the boundary equals ℓ_g.

Definition 31. Let O be an ontology, c a consequence, lab a labeling function, $S \subseteq O$ and $\ell_g \in L$ the *goal label*. The *modified assignment* lab_{S,ℓ_g} is given by

$$\mathsf{lab}_{S,\ell_g}(a) = \begin{cases} \ell_g, & \text{if } a \in S, \\ \mathsf{lab}(a), & \text{otherwise.} \end{cases}$$

A sub-ontology $S \subseteq O$ is called a *change set (CS)* for ℓ_g if the boundary for O,c under the labeling function lab_{S,ℓ_g} equals ℓ_g. It is a *minimal CS (MinCS)* if for every $S' \subset S$ we have that lab_{S',ℓ_g} does not equal ℓ_g.

Obviously, the original ontology O is always a change set for any goal label if $O \models c$. However, we are interested in performing minimal changes to the labeling function. Hence, we search for a change set of minimum cardinality. A change set of minimum cardinality, or *smallest CS* for short, is obviously also a MinCS. However, the reverse is not necessarily true. A MinCS is minimal with respect to set inclusion but is not necessarily a smallest CS since there might be several MinCS of different cardinality. This is similar to the minimality of MinA (see Definition 7), where a MinA is also not necessarily a MinA of minimum cardinality. It follows from [BPS07] that the problem of finding a smallest CS is NP-complete.

Let ℓ_g denote the goal label and ℓ_c the computed boundary for c. If $\ell_g \neq \ell_c$, we have three cases which are illustrated in Figure 4.2: either (1) $\ell_g < \ell_c$ (left), (2) $\ell_c < \ell_g$ (right), or (3) ℓ_g and ℓ_c are incomparable (middle). In our example with $\ell_c = \ell_3$, the three cases are given by ℓ_g being ℓ_0, ℓ_4, and ℓ_5, respectively. The sets L_c and L_g contain the user labels before and after the label changes respectively. Consider the first case, where $\ell_g < \ell_c$. Then, from Lemma 6 it follows that any diagnosis S is a change set for ℓ_g: since $\ell_g < \ell_c$, then for every diagnosis S', $\ell_g < \bigoplus_{a \in S'} \mathsf{lab}(a)$. But then, under the new labeling lab_{S,ℓ_g} we get that $\bigoplus_{a \in S} \mathsf{lab}_{S,\ell_g}(a) = \ell_g$. And hence, when the greatest lower bound of all $\bigoplus_{a \in S'} \mathsf{lab}_{S,\ell_g}(a)$ is computed, we obtain ℓ_g as a boundary. Using an analogous argument and Lemma 5, it is possible to show that if $\ell_c < \ell_g$, then every MinA is a change set for ℓ_g. The third case can be solved using a combination

of the previous two: if ℓ_g and ℓ_c are incomparable, we can first set as a partial goal $\ell'_g := \ell_g \otimes \ell_c$. Thus, we can first solve the first case, to set the boundary to ℓ'_g, and then, using the second approach, modify this new boundary once more to ℓ_g. Rather than actually performing this task as a two-step computation, we can simply compute a MinA and a diagnosis. The union of these two sets yields a CS. Unfortunately, the CS computed this way is not necessarily a MinCS, even if a smallest diagnosis or a smallest MinA is used, as shown in the following example.

Example 17. Let O,c and lab be as in Example 1 with the already introduced consequence $c : \mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$. We then know that $\ell_c := \ell_3$ is a boundary for O,c. Suppose now that c shall remain visible for those who see it already and additionally made available to customers, i.e. the goal label is $\ell_g := \ell_4$. Since $\ell_c < \ell_g$, we know that any MinA is a change set. Since all MinAs for O,c have exactly three elements, any change set produced this way will have cardinality three. However, $\{a_2\}$ is of cardinality one, is a subset of some of the MinAs and is still a CS. More precisely it is a MinCS.

To understand why the minimality of MinAs is not sufficient for obtaining a MinCS, we can look back to Lemma 5. This lemma says that in order to find a boundary, we need to compute the join of all λ_S, with S a MinA, and λ_S the meet of the labels of all axioms in S. But then, for any axiom $a \in S$ such that $\ell_g \leqslant \mathsf{lab}(a)$, modifying this label to ℓ_g will have no influence in the result of λ_S. In Example 17, there is a MinA $\{a_1, a_2, a_4\}$, where two axioms, namely a_1 and a_4 have a label greater or equal to $\ell_g = \ell_4$. Thus, the only axiom that needs to be relabeled is in fact a_2, which yields the MinCS $\{a_2\}$ shown in the example. Basically, we consider every axiom $a \in O$ such that $\ell_g \leqslant \mathsf{lab}(a)$ as *fixed* in the sense that it is superfluous for any change set. For this reason, we will deal with a generalization of MinAs and diagnoses, that we call IAS and RAS, respectively.

Definition 32 (IAS,RAS)**.** A *minimal inserted axiom set (IAS)* for ℓ_g is a subset $I \subseteq O_{\not\geqslant \ell_g}$ such that $O_{\geqslant \ell_g} \cup I \models c$ and for every $I' \subset I : O_{\geqslant \ell_g} \cup I' \not\models c$.

A *minimal removed axiom set (RAS)* for ℓ_g is a subset $R \subseteq O_{\not\leqslant \ell_g}$ such that $O_{\not\leqslant \ell_g} \setminus R \not\models c$ and for every $R' \subset R : O_{\not\leqslant \ell_g} \setminus R' \models c$.

Since there might be several IAS (RAS), computing all of them would allow to find a smallest one. The following theorem justifies the use of IAS and RAS when searching for a smallest change set.

Theorem 3. *Let ℓ_c be a boundary for O,c, ℓ_g the goal label, and m_R, m_I and m_U the cardinalities of a smallest RAS, a smallest IAS and a smallest union of an IAS and a RAS for ℓ_g, respectively. Then, for every IAS I and RAS R for ℓ_g it holds:*

- *if $\ell_g < \ell_c$ then R is a MinCS, if additionally $|R| = m_R$ then R is a smallest CS,*
- *if $\ell_c < \ell_g$ then I is a MinCS, if additionally $|I| = m_I$ then I is a smallest CS,*
- *if ℓ_c and ℓ_g are incomparable then $I \cup R$ is a MinCS, if additionally $|R \cup I| = m_U$ then $I \cup R$ is a smallest CS.*

The cardinality of a smallest union of an IAS and a RAS cannot be computed from the cardinalities of a smallest RAS and a smallest IAS. The reason is that combining the smallest IAS and RAS does not necessarily yield a smallest CS. The following example illustrates this.

Example 18. Assume $\{a_1, a_2\}, \{a_2, a_3\}$ are the smallest RAS and $\{a_1, a_4\}$ is the smallest IAS, then $\{a_1, a_2, a_4\}$ is the smallest CS and has cardinality 3. However, combining a smallest IAS and a smallest RAS might yield a MinCS (but not a smallest CS) of cardinality 4.

4.3.2 Computing a Smallest Change Set

In this section, we show how a smallest CS can be computed. We first present the most obvious approach that is based in the computation of all MinAs and diagnoses. Afterward, we show how this idea can be improved by considering fixed portions of the ontology, as described before. We further improve this approach by showing that it usually suffices to compute only partial MinCS by putting a cardinality limit, thus reducing the search space and execution time of our method.

Using Full Axiom Pinpointing

Although we have shown in Example 17 that MinAs and diagnoses do not yield MinCS or even smallest CS directly, both of these change sets can still be deduced from the set of all MinAs and diagnoses, as shown by the following lemma.

Lemma 8. *Let I (R) be an IAS (RAS) for ℓ_g, then there is a MinA (diagnosis) S such that $I = S \setminus O_{\geqslant \ell_g}$ ($R = S \setminus O_{\leqslant \ell_g}$).*

Lemma 8 shows that we can compute the set of all IAS by first computing all MinAs and then removing the set of fixed elements $O_{\geqslant \ell_g}$ from it. Thus, the most naïve approach for computing a change set of minimum cardinality is to first find all MinAs, then compute the set of all IAS by removing all elements in $O_{\geqslant \ell_g}$, and finally search for the IAS having the least elements. The same procedure applies respectively to RAS.

As explained in sections above, the task of computing all MinAs, also called axiom pinpointing, has been widely studied in recent years, and there exist black-box implementations based on the HST algorithm [Kal+07; Sun08]. The HST algorithm has been explained in Section 4.2.4 already in detail. The HST algorithm makes repeated calls to an auxiliary procedure that computes a single MinA. Further MinAs are found by building a tree, where nodes are labeled with MinAs. If the MinA labeling a node has n axioms ($S := \{a_1, \ldots, a_n\}$), then this node will have n children: the i-th child is labeled with a MinA obtained after removing a_i from the ontology. This ensures that each node is labeled with a MinA distinct from those of its predecessors. Although not stated explicitly in the axiom pinpointing literature, it is clear that the same HST algorithm can be used for computing all diagnoses. The only variant necessary is to have a subroutine capable of computing one such diagnosis, which can be obtained by dualizing the algorithm computing one MinA (see Algorithm 4.5 for an example on

how this dualization works). In our experiments, we used this approach as a basis to measure the improvement achieved by the optimizations that will be introduced next.

Using Fixed Axioms and Cardinality Limit

Naïvely a smallest CS can be found by computing all MinCS and selecting a smallest. As explained above, the task of computing all MinCS is related to computing all diagnoses and all MinAs, which has been widely studied in recent years, and there exist black-box implementations based on the HST algorithm. Our approach to compute a smallest CS follows similar ideas. The HST algorithm makes repeated calls to an auxiliary procedure that computes a single MinCS. Further MinCS are found by building a tree, where nodes are labeled with MinCS and edges with axioms. If the MinCS labeling a node has n axioms ($S := \{a_1, \ldots, a_n\}$), then this node will have n children: the edge to the i-th child labeled with a_i, the child labeled with a MinCS that is allowed to contain neither a_i nor any ancestor's edge label. This ensures that each node is labeled with a MinCS distinct from those of its predecessors.

For the auxiliary procedure to compute a single MinCS, we will use two sub procedures extracting RAS and IAS, respectively. In Algorithm 4.4 we present a variation of the logarithmic MinA extraction procedure presented in [BS08] that is able to compute an IAS or stop once this has reached a size n and return a partial IAS. We also show the RAS variant in Algorithm 4.5. Given a goal label ℓ_g, if we want to compute a IAS or a partial IAS of size exactly n for a consequence c, then we would make a call to extract-partial-IAS$(O_{\geqslant \ell_g}, O_{\not\geqslant \ell_g}, c, n)$. Similarly, a call to extract-partial-RAS$(O_{\not\leqslant \ell_g}, O_{\not\leqslant \ell_g}, c, n)$ yields a RAS of size $\leqslant n$ or a partial RAS of size exactly n. The cardinality limit will be used to avoid unnecessary computations when looking for a smallest CS.

Given the procedures to extract RAS and IAS, Algorithm 4.6 extracts a MinCS. In order to label a node, we compute a MinCS with extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g, H, n)$, where H is the set of all labels attached to edges on the way from the node to the root of the tree. Note that axioms in H are removed from the search space to extract the IAS and RAS. Furthermore, axioms in the IAS are considered as *fixed* for the RAS computation. The returned set is a MinCS of size $\leqslant n$ or a partial MinCS of size n.

Example 19. Returning to our running example, suppose now that we want to hide c from development engineers and make it available to customers, i.e. modify the label of consequence c to $\ell_g = \ell_5$. Algorithm 4.6 starts by making a call to extract-partial-IAS$(O_{\geqslant \ell_5}, O_{\not\geqslant \ell_5}, c, \infty)$.[6] A possible output for this call is $I = \{a_3\}$. We can then call extract-partial-RAS$(O_{\not\leqslant \ell_5} \setminus I, O_{\not\leqslant \ell_5} \setminus I, c, \infty)$, which may output e.g. the set $R = \{a_1\}$. Thus, globally the algorithm returns $\{a_3, a_1\}$, which can be easily verified to be a MinCS for ℓ_5.

One of the advantages of the HST algorithm is that the labels of any node are always ensured not to contain the label of any of its predecessor nodes. In particular this means that even if we compute a partial MinCS, the algorithm will still correctly find all MinCS that do not contain any of the partial MinCS found during the execution.

[6] For this example, we ignore the cardinality limit, as we want to find only one MinCS.

Algorithm 4.4 Compute a (partial) IAS

Procedure extract-partial-IAS($O_{\mathsf{fix}}, O_{\mathsf{test}}, c, n$)
Input: O_{fix}: fixed axioms; O_{test}: axioms; c: consequence; n: limit
Output: first n elements of a minimal $S \subseteq O_{\mathrm{test}}$ such that $O_{\mathrm{fix}} \cup S \models c$

1: **Global** $l := 0, n$
2: **return** extract-partial-IAS-r($O_{\mathsf{fix}}, O_{\mathsf{test}}, c$)

Subprocedure extract-partial-IAS-r($O_{\mathsf{fix}}, O_{\mathsf{test}}, c$)

1: **if** $n = l$ **then**
2: **return** $\emptyset$
3: **if** $|O_{\mathrm{test}}| = 1$ **then**
4: $l := l + 1$
5: **return** O_{test}
6: $S_1, S_2 := \mathsf{halve}(O_{\mathrm{test}})$ (partition O_{test} so that $||S_1| - |S_2|| \leqslant 1$)
7: **if** $O_{\mathrm{fix}} \cup S_1 \models c$ **then**
8: **return** extract-partial-IAS-r(O_{fix}, S_1, c)
9: **if** $O_{\mathrm{fix}} \cup S_2 \models c$ **then**
10: **return** extract-partial-IAS-r(O_{fix}, S_2, c)
11: $S_1' :=$ extract-partial-IAS-r($O_{\mathrm{fix}} \cup S_2, S_1, c$)
12: $S_2' :=$ extract-partial-IAS-r($O_{\mathrm{fix}} \cup S_1', S_2, c$)
13: **return** $S_1' \cup S_2'$

Algorithm 4.5 Compute a (partial) RAS

Procedure extract-partial-RAS($O_{\mathsf{nonfix}}, O_{\mathsf{test}}, c, n$)
Input: O_{nonfix}: axioms; $O_{\mathrm{test}} \subseteq O_{\mathrm{nonfix}}$: axioms; c: consequence; n: limit
Output: first n elements of a minimal $S \subseteq O_{\mathrm{test}}$ such that $O_{\mathrm{nonfix}} \setminus S \not\models c$

1: **Global** $l := 0, O_{\mathrm{nonfix}}, n$
2: **return** extract-partial-RAS-r($\emptyset, O_{\mathsf{test}}, c$)

Subprocedure extract-partial-RAS-r($O_{\mathsf{hold}}, O_{\mathsf{test}}, c$)

1: **if** $n = l$ **then**
2: **return** $\emptyset$
3: **if** $|O_{\mathrm{test}}| = 1$ **then**
4: $l := l + 1$
5: **return** O_{test}
6: $S_1, S_2 := \mathsf{halve}(O_{\mathrm{test}})$ (partition O_{test} so that $||S_1| - |S_2|| \leqslant 1$)
7: **if** $O_{\mathrm{nonfix}} \setminus (O_{\mathrm{hold}} \cup S_1) \not\models c$ **then**
8: **return** extract-partial-RAS-r($O_{\mathrm{hold}}, S_1, c$)
9: **if** $O_{\mathrm{nonfix}} \setminus (O_{\mathrm{hold}} \cup S_2) \not\models c$ **then**
10: **return** extract-partial-RAS-r($O_{\mathrm{hold}}, S_2, c$)
11: $S_1' :=$ extract-partial-RAS-r($O_{\mathrm{hold}} \cup S_2, S_1, c$)
12: $S_2' :=$ extract-partial-RAS-r($O_{\mathrm{hold}} \cup S_1', S_2, c$)
13: **return** $S_1' \cup S_2'$

Algorithm 4.6 Compute a (partial) MinCS

Procedure extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g, H, n)$

1: **return** extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g,$
$\ell_g \not\preccurlyeq \mathsf{lbl}(c) \wedge O_{\not\succcurlyeq \ell_g} \not\models c,$
$\ell_g \not\succcurlyeq \mathsf{lbl}(c) \wedge O_{\not\preccurlyeq \ell_g} \models c,\ H, n)$

Procedure extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g, \mathsf{is}_I, \mathsf{is}_R, H, n)$

Input: O, lab: labeled ontology; c: consequence; ℓ_g: goal label; is_I: decision to compute IAS; is_R: decision to compute RAS; H: HST edge labels; n: limit

Output: first n elements of a MinCS $S \subseteq O$

1: **if** $\mathrm{is}_I \wedge O_{\not\succcurlyeq \ell_g} \cup (O_{\not\succcurlyeq \ell_g} \setminus H) \not\models c$ or $\mathrm{is}_R \wedge H \models c$ **then**
2: **return** $\emptyset$ (HST normal termination)
3: **if** is_I **then**
4: $I :=$ extract-partial-IAS$(O_{\not\succcurlyeq \ell_g}, O_{\not\succcurlyeq \ell_g} \setminus H, c, n)$
5: **if** is_R and $O_{\not\preccurlyeq \ell_g} \setminus I \models c$ **then**
6: $R :=$ extract-partial-RAS$(O_{\not\preccurlyeq \ell_g} \setminus I, O_{\not\preccurlyeq \ell_g} \setminus (I \cup H), c, n - |I|)$
7: **return** $I \cup R$

Since we are interested in finding the MinCS of minimum cardinality, we can set the limit n to the size of the smallest CS found so far. This limit is initially fixed to the size of the ontology. If extract-partial-MinCS outputs a set with fewer elements, we are sure that this is indeed a full MinCS, and our new smallest known CS. The HST algorithm will not find all MinCS in this way, but we can be sure that one MinCS with the minimum cardinality will be found. The idea of limiting cardinality for finding a smallest MinCS can be taken a step further by not expanding each node for all the axioms in it, but rather only on the first $n - 1$, where n is the size of the smallest CS found so far. This further reduces the search space by decreasing the branching factor of the search tree. Notice that the highest advantage of this second optimization appears when the HST is constructed in a depth-first fashion. In that case, a smaller MinCS found further below in the tree will reduce the branching factor of all its predecessors. So the cardinality limit reduces the search space in two dimensions: (1) the computation of a single MinCS is limited to n axioms and (2) only $n - 1$ axioms are expanded from each node. Algorithm 4.7 is the resulting HST algorithm. The following theorem shows that it is correct.

Theorem 4. *Let O be an ontology, c a consequence with $O \models c$, and ℓ_g a goal label. If m is the minimum cardinality of all CS for ℓ_g, the Algorithm 4.7 outputs a CS S such that $|S| = m$.*

Proof. The described algorithm outputs a CS since the globally stored and finally returned S is only modified when the output of extract-partial-MinCS has size strictly smaller than the limit n, and hence only when this is indeed a CS itself. Suppose now that the output S is such that $m < |S|$, and let S_0 be a MinCS such that $|S_0| = m$, which exists by assumption. Then, every set obtained by calls to extract-partial-MinCS has size strictly greater than m, since otherwise, S and n would be updated. Consider now an arbitrary set S' found during the execution through a call to extract-partial-

Algorithm 4.7 Compute a smallest CS by a HST algorithm

Procedure HST-extract-smallest-CS$(O, \mathsf{lab}, (L, \leqslant), c, \ell_g)$
Input: O,lab: labeled ontology; $(L, \leqslant)$: lattice; c: consequence; ℓ_g: goal boundary
Output: a smallest CS S

1: **Global** $\mathbf{C}, \mathbf{H}, S := O, n := |O|, c,$
   $\mathrm{is}_I := \ell_g \not\leqslant \mathsf{lbl}(c) \wedge O_{\geqslant \ell_g} \not\models c;$
   $\mathrm{is}_R := \ell_g \not\geqslant \mathsf{lbl}(c) \wedge O_{\not\leqslant \ell_g} \models c$
2: expand-HST-CS$(\emptyset)$
3: **return** S

Procedure expand-HST-CS(H)
Input: H: list of edge labels
Side effects: modifications to $\mathbf{C}$ and $\mathbf{H}$

1: **if** there exists some $H' \in \mathbf{H}$ such that $H' \subseteq H$ **or**
      H' contains a prefix-path P with $P = H$ **then**
2:    **return** (early termination $\otimes$)
3: **else if** there exists some $Q' \in \mathbf{C}$ such that $H \cap Q' = \emptyset$ **then**
4:    $Q := Q'$ (MinCS reuse)
5: **else**
6:    $Q :=$ extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, H, n)$
7:    **if** $\emptyset = Q$ **then**
8:       $\mathbf{H} := \mathbf{H} \cup \{H\}$ (normal termination $\odot$)
9:       **return**
10:   **if** $|Q| < |S|$ **then**
11:      $n := |Q|$
12:      $S := Q$
13:   $\mathbf{C} := \mathbf{C} \cup \{Q\}$
14: **for** the first $(n-1)$ axioms $a \in Q$ **do**
15:   expand-HST-CS$(H \cup \{a\})$

MinCS, and let $S'_n := \{a_1, \ldots, a_n\}$ be the first n elements of S'. Since S' is a (partial) MinCS, it must be the case that $S_0 \not\subseteq S'_n$ since every returned MinCS is minimal in the sense that no axiom might be removed to obtain another MinCS. Then, there must be an $i, 1 \leqslant i \leqslant n$ such that $a_i \notin S_0$. But then, S_0 will still be a MinCS after axiom $\{a_i\}$ has been removed. Since this argument is true for all nodes, it is in particular true for all leaf nodes, but then they should not be leaf nodes, since a new MinCS, namely S_0 can still be found by expanding the HST, which contradicts the fact that S is the output of the algorithm. □

Example 20. Returning to our running example, suppose that we want to hide c from development engineers, i.e. set the label of c to $\ell_g = \ell_0$. Algorithm 4.6 first calls extract-partial-RAS$(O_{\not\leqslant \ell_0}, O_{\not\leqslant \ell_0}, c, 5)$. A possible output of this call is $R = \{a_2, a_3\}$. The tree now branches through a_2 and a_3. In the first case it calls extract-partial-RAS$(O_{\not\leqslant \ell_0}, O_{\not\leqslant \ell_0} \setminus \{a_2\}, c, 2)$, which could yield $R = \{a_4, a_5\}$. This might be a partial MinCS since its size equals the cardinality limit. The next call extract-partial-

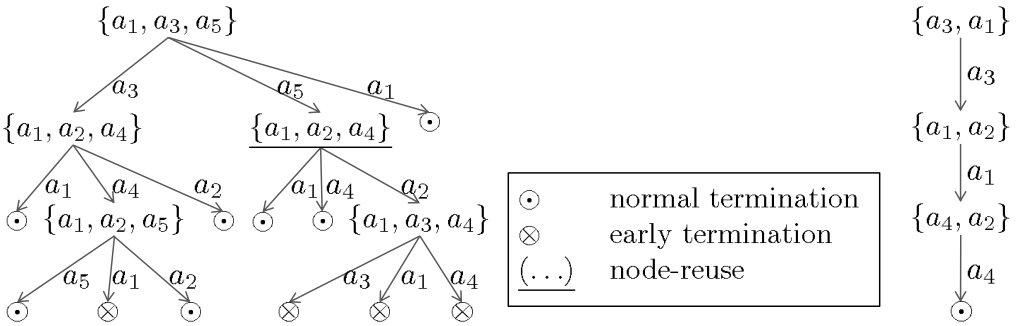

Figure 4.3: Hitting Set Trees to compute all MinAs (left) and a smallest change set for $\ell_g = \ell_5$ (right)

$\mathsf{RAS}(O_{\not\leqslant \ell_0}, O_{\not\leqslant \ell_0} \setminus \{a_2, a_4\}, c, 2)$ yields a smallest $R = \{a_1\}$, and the HST terminates. Notice that if $\{a_1\}$ had been the first MinCS found, the process would have immediately terminated.

Efficient implementations of the original version of the HST algorithm rely on several optimizations. Two standard optimizations described in the literature are node-reuse and early path termination (see, e.g. [Kal+07; Sun08; BKP09b]). Node-reuse keeps a history of all nodes computed so far in order to avoid useless (and usually expensive) calls to the auxiliary procedure that computes a new node. Early path termination, on the other hand, prunes the Hitting Set Tree by avoiding expanding nodes when no new information can be derived from further expansion. In order to avoid unnecessary confusion, we have described the modified HST algorithm without including these optimizations. However, it should be clear that both, node-reuse and early path termination, can be included in the algorithm without destroying its correctness. The implementation used for our experiments contain these two optimizations.

Example 21. We continue Example 1 with the already introduced consequence c : *ServiceWithComingPriceIncrease*(*ecoCalculatorV1*). For goal label $\ell_g = \ell_5$, Figure 4.3 shows the expansion of the HST trees computing all MinAs and all diagnoses (left), in comparison with the one obtained for computing a smallest change set with both optimizations, fixed axioms and cardinality limit (right). Obviously, the number of nodes, the node cardinality and the number of tree expansions is lower.

Parts of this sections result were published in [KP10a] and continued in [KP10b]. In difference to the presentation here, [KP10a] had not one HST Algorithm to find a smallest CS but two HST Algorithms to find a smallest IAS and a smallest RAS separately. The variant presented here is guaranteed to find the smallest CS, as given in the proof above. In contrast to that, combining the smallest IAS and RAS does not necessarily yield a smallest CS, as the following example shows. Assume $\{a_1, a_2\}, \{a_2, a_3\}$ are the smallest RAS and $\{a_1, a_4\}$ is the smallest IAS, then $\{a_1, a_2, a_4\}$ is the smallest CS, but choosing a smallest IAS and a smallest RAS might yield a MinCS (but not a smallest CS) of cardinality 4. In [KP10a] we also investigated the performance gain by taking not only advantage of fixing a subset of the axioms and limiting cardinality

but also by taking the labels of the remaining axioms into account. Since this yielded no significant performance gain, presentation is omitted here.

An empirical evaluation of our algorithms from this section is provided in Section 6.2.2.

4.4 Conclusions of the Chapter

We have considered a scenario where ontology axioms are labeled and user labels determine views on the ontology, i.e., sub-ontologies that are obtained by comparing the user label with the axiom labels. The labeled axioms entail consequences, but their label is not explicitly given. However, intuitively a consequence should be as access restricted as the axioms from which it follows. We introduced an approach formalizing this intuition and showed how to compute a label for a given consequence, called boundary. Our approach can be used for large-scale ontologies since, on the one hand, it allows to pre-compute consequences without having to do this separately for all possible views. Once we have computed a boundary for the consequence, checking whether this consequence follows from a sub-ontology is reduced to a simple label comparison. On the other hand, the fact that we employ a black-box approach for computing the boundary allows us to use existing highly-optimized reasoners, rather than having to implement a new reasoner from scratch.

Our general framework allows using any restriction criterion that can be represented using a lattice, such as user roles, levels of trust, granularity, or degrees of certainty. With user roles, each axiom label defines the user roles able to see the axiom and each user label defines the sub-ontology containing the axioms visible to this user. In the presence of trust restrictions, the user label specifies the trust level required for the ontology axiom. This supports scenarios with axioms from different sources, like company-internal with high trust level and public Web with low trust level. In the presence of uncertainty, e.g. in possibilistic reasoning, each axiom has an associated certainty degree in the interval $[0, 1]$. The user label then specifies the certainty degree required for the axioms and the consequences. Similarly, granularity restrictions (i.e., on how much details the ontology should provide for the user) can be expressed by a total order.

Our framework is independent of a specific reasoner. To stay as general as possible, we do not fix a specific ontology language. We just assume that ontologies are finite sets of axioms such that every subset of an ontology is again an ontology and we require the consequence relation to be monotonic. The elements of the labeling lattice which are used as user label have to be join-prime.

The security administrator might not be satisfied with the access restriction level computed for a consequence from the access restriction levels of the axioms that entail the consequence. In this case, the computed boundary is different from the intended label and a repair of the consequence's boundary is required. This is only indirectly possible by changing some axiom labels. Based on ontology repair techniques we developed algorithms to compute a change set of minimal cardinality, which contains axioms to be relabeled in order to yield a desired boundary for a given consequence. The base problem, finding a smallest MinA and diagnosis without computing all of

them might be interesting beyond our application domain. Our algorithms take advantage of (1) fixing a subset of the axioms which are known not to be part of the search space and (2) limiting cardinality of MinCS to be computed in the Hitting Set Tree to the size of the smallest known change set. All our algorithms are black-box based, which means that they can be used with any off-the-shelf reasoner, without the need for modifications.

Revisiting the research questions from Section 1.4, these results answer the questions 1, 2 and 3.

5 User Support to Assign Access Restrictions to Ontology Axioms

Instead of labeling axioms one by one, which might be unfeasible for large ontologies, labels could be assigned to groups of axioms and consequences that share defined commonalities. Also if security administrators are unfamiliar with the task of assigning access restrictions to ontology axioms and consequences, it might make sense to reduce this task to a better known task.

A simple approach for axioms only is to transfer a document's access restrictions to the document ontology, i.e. all contained axioms. The case where one axiom is contained in multiple document ontologies needs to be handled appropriately. The intuition is that such an axiom is as visible as the most publicly readable document allows.

A more sophisticated approach for consequences, which includes axioms, is a query-based assignment of access restrictions. All responses to that query in the form of a set of consequences shall receive a defined label. Many queries with different goal labels can be combined, which results in a set of consequences with different goal labels. Enforcing those labels is an extension of label repair for a single consequence from Section 4.3 since again every consequence label can only be changed by changing axiom labels. In the case of multiple goals, there might arise conflicts requiring an appropriate resolution strategy. Furthermore, we compare axiom labeling and consequence label repair to a completely different alternative of restricting access to an ontology, namely query rewriting. While both strategies keep knowledge secret which is intended to be secret, we observe different availability of knowledge which could be made available without telling secrets.

Parts of this chapter have been published in [KS08; KS10].

5.1 Document-Based Access Restrictions

This section describes a simple approach of applying access restrictions from documents to ontology axioms. This is especially helpful for security administrators familiar with access restrictions to documents in a file system but not (yet) with access restrictions to ontology axioms.

The architecture of a access restricted semantic document store in Figure 1.7 contains a document ontology for each document. A basic intuition could be to assign an access label to all axioms contained in a document ontology, which corresponds to the document access restrictions.

All document ontologies might be integrated into one large ontology in order to make all joint implicit knowledge explicit with a reasoner and for answering queries on the complete knowledge, both with respect to the user's permissions. In this case the ontology axioms preserve the intended level of visibility. It could happen that one and the same axiom comes from multiple document ontologies. In this case the axiom requires a defined joint label representing the least restrictive of all documents' access restrictions.

Given are an RBAC matrix and a set of documents and corresponding document ontologies. An appropriate labeling lattice can be computed from the RBAC matrix with methods from Section 3.2. An important requirement for the labeling lattice is, as discussed in Section 4.2.3, that all user labels are join prime. It is important also here because the join of multiple labels has to be computed when an axiom is contained in several document ontologies. In case the obtained lattice does not fulfill this, it has to be changed, for example by adding new lattice elements, while preserving the order among the user labels. For example, a labeling lattice for the strict interpretation of the RBAC matrix in Table 1.1 is depicted in Figure 1.3.

The problem is to find a label assignment for the axioms in the document ontologies that reflects the access restrictions to the respective documents.

A solution with the intuition described above could be the following. Each user role with read permission has a corresponding user label in the labeling lattice. The supremum of those lattice elements yields the axiom label. If the axiom appears in multiple document ontologies, the supremum of those lattice elements for all the documents yields the axiom label. In the following we give an example.

Example 22. We extend the example scenario from Section 1.2. In addition to the document types given in the RBAC matrix in Table 1.1, we introduce a requirements document (RQD) readable by DE, ME, TE, SV, LDE, CSE, CU. We discussed this extension of the matrix in Section 3.2 already, and explained that for the strict interpretation of this extended RBAC matrix the right part of Figure 3.6 depicts the concept lattice. This lattice is a valid labeling lattice, since all elements representing user roles are join prime. We leave the set of axioms of the scenario unchanged, but assume they do not have a label assigned yet.

For this extended scenario, the Table 5.1 provides the result of a document-based access restriction assignment for each ontology axiom. For each row, the first column contains the axiom, which needs a label assigned. The second column lists the document types in whose corresponding document ontologies the axiom is contained. For

Axiom	Document types	User roles	Axiom label
a_1	{UM, RQD, DD, CCD}	{DE, ME, TE, SV, LDE, CSE, CU, SP}	ℓ_1
a_2	{IG, DD}	{DE, ME, TE, SV, LDE, CSE, SP}	ℓ_2
a_3	{DD}	{DE, ME, TE, SV, LDE, CSE}	ℓ_3
a_4	{RQD}	{DE, ME, TE, SV, LDE, CSE, CU}	ℓ_4
a_5	{CCD}	{SV, LDE, CSE, CU}	ℓ_5

Table 5.1: Axioms and their containment in document ontologies

example, the fact that the service called "$ecoCalculatorV1$" is an $EUecoService$ and a $Highperformance Service$ has been stated in the documents of type user manual, requirements document, design document and customer contract document. The axiom a_1 represents this fact formally and is contained in the corresponding document ontologies. The third column lists the user roles that are allowed to read at least one of the document types. The fourth column contains the supremum of the lattice elements representing those user roles. This is the label assigned to the axiom. For example, axiom a_1 receives the label ℓ_1.

The described procedure assigns a label to an axiom with respect to document permissions and with respect to the set of documents for which the axiom formally represents knowledge.

Beyond that, the procedure can in fact unveil wrong permissions in the RBAC matrix if it turns out that some inferable knowledge is more public than intended, i.e. the boundary of a consequence is higher than intended. This might easily happen, if the RBAC matrix has been designed in order to protect the explicit information in the documents and the explicit knowledge in the document ontologies without considering logical consequences that follow. Methods from Section 4.3 to repair a consequence label might be useful in that case.

5.2 Query-Based Access Restrictions

A query can be used to retrieve explicit and implicit knowledge, i.e. a set of consequences from a given ontology. In this section we use a query to address a subset of consequences that shall receive a defined access restriction. For the security administrator, a query-based assignment of access restrictions might be more convenient than assigning axiom labels one by one to individual axioms. Such a query could, e.g., address knowledge about a concept and all subconcepts in order to restrict knowledge along the subsumption hierarchy or restrict knowledge along a selected object property as it has been proposed, e.g., in [IO05; QA03]. This is comparable to information systems restricting access to files in a directory and all subdirectories. Conflict resolution mechanism might be necessary then since, e.g., a concept might have multiple superconcepts.

In Chapter 4 we introduced an approach to enforce access restrictions by labeled ontologies, which we call *label filtering* in the following. We additionally discuss an

alternative approach to enforce access restrictions called *query rewriting*. The label filtering approach assumes an a priori labeling of axioms in the ontology to consistently derive labels for implicit consequences. Axioms and consequences are delivered to a user based on a comparison of user label and axiom label. The query rewriting approach proposed in [CS09] is based on the idea of rewriting user queries based on the role a user has in such a way that the result to the rewritten query only returns knowledge the user is allowed to see.

Our assessment of label filtering vs. query rewriting concludes that while both approaches prevent uncovering secret knowledge, label filtering is more complete in the sense that it does not suppress knowledge the user is allowed to see while this happens frequently in query rewriting. Furthermore, label filtering is independent of the ontology language. However, label filtering requires an a priori labeling of axioms and it is not clear from previous work how to create an access labeling from query-based access restrictions. Based on algorithms from Section 4.3 to repair a single consequence label we present algorithms to repair a set of consequence labels with multiple optimizations. As discussed in Section 4.3, changing a consequence label is only possible by changing labels of a set of axioms and identifying this set of axioms is in general as hard as finding an explanation and a diagnosis for a consequence.

The main results from this section are (1) a comparison of label filtering vs. query rewriting, (2) algorithms to repair a given axiom labeling in an optimal way so that a query-based access restriction is enforced to explicit and implicit knowledge, (3) conflict resolution strategies for cases where query-based access restrictions contain conflicts. In the empirical evaluation in Section 6.2.3 we report experiments on real-world data showing that a significant number of results are retained using the label filtering method.

5.2.1 Access Restrictions as Queries

The set of all concept and role names occurring in the axioms of an ontology O is called the signature $sig(O)$ of the ontology. A query to an ontology is a conjunction $Q = A_1, \ldots, A_n$ of OWL axioms over $sig(O)$, but not necessarily from O, containing variables. For a concrete definition of the form of axioms see [SP07]. The set of variables occurring in Q is denoted as $var(Q)$. Let $ind(O)$ be the set of individuals in O, then the result of a query is the set of all mappings $\mu : var(Q) \rightarrow ind(O)$ assigning individuals from O to variables in Q. An answer $\mu(Q)$ to a query Q is an instantiation of all variables in the query, so that $O \models \mu(Q)$ [SP07]. Note that there might be several possible μ for one query.

Assume we have two user roles: customers and employees and we want both to be able to query some of the axioms of the ontology given in Example 1. Employees shall have full access but customers shall not see if any product gets an increased price soon. This restriction could be defined by enumerating all query responses except the price increase as permissions and assigning them to the respective user role. There are two problems with this approach. First of all, the price increase can still be inferred if the axioms of O can be queried. Further, enumerating all query responses, however, is

not feasible in practice and asks for more efficient ways of specifying these restrictions, e.g. by means of a query.

A way is to define permissions intentionally in terms of queries over the signature of the ontology. More specifically, we can describe facts that should not be accessible by a defined user role in terms of a query. In the case of the example above, for customers we could formulate the restriction

$$\mathit{ServiceWithComingPriceIncrease}(x)$$

stating that for no instantiation of the variable x it should be possible to infer that it is an instance of *ServiceWithComingPriceIncrease*.

5.2.2 Query Rewriting vs. Label Filtering

There are different ways for implementing access control for ontological knowledge. While query rewriting extends a user's query to include all access restrictions, label filtering only allows a subset of the ontology to be used to answer the unchanged query.

Access Control by Query Rewriting

One option for enforcing access restrictions is by means of query rewriting. This approach has been proposed in [CS09] as a suitable way for enforcing access restrictions in the context of SPARQL queries, while the TBox is assumed to be completely public. Similar approaches are also allowing to hide TBox parts [GH08; BSH07] or to define not the restrictions but the permissions by a query [Cal+08]. The idea in [CS09] is to automatically add filter conditions to the query that suppress such answers the user is not supposed to see. Given a Query Q and a set of access restrictions $\{AR_1, \cdots, AR_n\}$ that apply to the current user, the query can be rewritten to a new query that is defined as

$$Q \wedge \neg AR_1 \wedge \cdots \wedge \neg AR_n$$

where the junction of two queries $Q_1 \wedge Q_2$ is the junction of all contained query axioms $\bigwedge_{q \in Q_1} q \wedge \bigwedge_{q \in Q_2} q$ [SP07]. This way of rewriting the query based on the access restrictions of the individual users effectively prevents the system from giving away restricted knowledge. In particular, using query rewriting, the effective answer to a query is

$$\{\mu(Q) | O \models \mu(Q \wedge \neg AR_1 \wedge \cdots \wedge \neg AR_n)\}$$

It however presents a problem: it hides more knowledge than necessary. In particular, in the example above where we want to hide from customers that some product is increased in price, the query rewriting approach hides too much knowledge. If a customer for instance asks the system for all high performance services, thus $Q = \mathit{HighperformanceService}(x)$, this query will be rewritten to an extended query $\mathit{HighperformanceService}(x) \wedge \neg \mathit{ServiceWithComingPriceIncrease}(x)$. This query will only return high performance services which will not be increased in price. These do not exist by definition, so the response is empty. This is unfortunate, because the knowledge that *ecoCalculatorV1* is a high performance service was not supposed to be

hidden. It has to be hidden in this approach, because it does not allow to hide parts of the TBox. Similarly querying for instances of the remaining four concept names in $sig(O)$ are filtered, resulting in five queries without an answer.

Access Control by Label Filtering

The framework to control access to an ontology's axioms has been introduced in Chapter 4. In contrast to the query rewriting approach, the TBox is not assumed to be completely public with the label filtering approach.

Applied to our scenario, with the user roles customer (user label ℓ_C) and employee (user label ℓ_E), let the labeling lattice be $(L, \leqslant)$ with $L := \{\ell_C, \ell_E\}$ and $\leqslant := \{(\ell_E, \ell_C)\}$. Let the labeling function lab assign label ℓ_C to axioms a_1, a_2, a_3 and label ℓ_E to axioms a_4, a_5. Employees can see $O_{\geqslant \ell_E} = \{a_1, a_2, a_3, a_4, a_5\}$, i.e. the complete ontology. Customers can see $O_{\geqslant \ell_C} = \{a_1, a_2, a_3\}$. The boundary of the consequence c_1 is ℓ_E, as can be easily verified, which means that employees can see it but customers cannot. Consequence c_2 has boundary ℓ_C, i.e. employees and customers can see it. Apart from c_1, c_2, the instance relationships to the three remaining concepts in $sig(O)$ have the boundary ℓ_C. A customer querying for instances of the five concept names in the ontology will receive no answer for $Q = \mathit{ServiceWithComingPriceIncrease}(x)$ but will receive an answer for the four remaining queries. So label filtering provides 4/5 answers, while query rewriting provides 0/5 answers.

Discussion

As we have seen, query rewriting and label filtering are approaches of ensuring that no classified knowledge is given to users that do not have the permission to see it. Both approaches do neither require tracking the history of queries nor prohibiting query askers of the same user role to share knowledge. We have seen that query rewriting is suboptimal with respect to availability in the sense of preserving maximal access to non-restricted knowledge. Label filtering provides a higher availability and is more general since it is independent of the concrete ontology language which makes the approach preferable in many situations. However, it requires an a priori axiom labeling, and it is not clear how to enforce query-based access restrictions. The Chapter 4 on labeled ontologies focused on computing a consequence's label based on axiom labels (see Section 4.2) and on repairing the axiom labeling in order to determine one consequence's label (see Section 4.3). However, access restrictions in the form of queries might require changing labels of multiple consequences simultaneously. Such a mechanism will be presented in the next section. Our main quality criterion for the algorithms is availability. In the empirical evaluation in Section 6.2.3 we measure how many knowledge is additionally accessible with label filtering compared to query rewriting.

5.2.3 Repairing Access Restrictions to Multiple Consequences

In the last section we have only shown that there is an axiom labeling to enforce access restrictions for a selected example. Now we will elaborate how to compute it in general.

We are starting from an arbitrary label assignment, and change it in a minimal way so that a given access restriction is enforced.

Example 23. We continue Example 1. The boundaries of the consequences c_1 and c_2 can be computed with the methods provided in Section 4.2. The computed boundary of the consequence c_1 is ℓ_3, since $= (\ell_1 \otimes \ell_2 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_2 \otimes \ell_5) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_5)$. The computed boundary of the consequence c_2 is ℓ_2, since $= (\ell_1 \otimes \ell_2) \oplus (\ell_1 \oplus \ell_3)$. For users ℓ_0 and ℓ_3, consequences c_1 and c_2 are visible. For user ℓ_2, only c_2 is visible.

We now define a notion for changing an axiom label assignment. We use the notion of function lbl from Definition 29, in order to refer to the computed boundary $\mathsf{lbl}(c)$ for a given consequence c.

Definition 33 (MCS)**.** Let O be an ontology, c a consequence following from O, $(L, \leqslant)$ a lattice, lab a labeling function, G a set of goals of the form (c, ℓ_g) with goal label ℓ_g for consequence c, M a set of assignments (a, ℓ) of label ℓ to axiom a, where each axiom a appears only once. The *modified assignment* lab_M is defined to be

$$\mathsf{lab}_M(a) = \begin{cases} \ell, & \text{if } (a, \ell) \in M, \\ \mathsf{lab}(a), & \text{otherwise.} \end{cases}$$

The respective consequence labeling function lbl_M based on lab_M is given by Definition 29. The set M is called *minimal multiple change set (MCS)* iff for any $c, (c, \ell_g) \in G : \mathsf{lbl}_M(c) = \ell_g$ and for every $M' \subset M$ there is at least one goal $(c, \ell_g) \in G$ with $\mathsf{lbl}_{M'}(c) \neq \ell_g$.

Whether we can find a lab_M fulfilling a given goal set is independent of the label assignment lab we start from. For default deny-all behavior, we start with all axioms assigned to the bottom lattice element. For default allow-all behavior, we start with all axioms assigned to the top lattice element. We will now introduce the computation of a smallest change set for one goal and building on that introduce the computation of a smallest MCS.

Computing a Smallest Change Set For One Goal Label

If G is the singleton set of only one tuple (c, ℓ), computing a MCS boils down to computing a MinCS which has been introduced in Section 4.3. For every MinCS $S \subseteq O$ there is a MCS $M := \{(a, \ell_g) \mid a \in S\}$ and $\mathsf{lbl}_M(c) = \ell_g$ holds. The computation of a MinCS exploited main ideas from axiom-pinpointing [Kal+07; BP10b] and we presented a black-box approach that yields the desired set. Intuitively, a consequence c needs to be made more public if $\ell_g > \mathsf{lbl}(c)$ or less public if $\ell_g < \mathsf{lbl}(c)$. From the perspective of the target users who see $O_{\geqslant \ell_g}$, the former is achieved by including an axiom set IAS to their ontology and the latter by removing an axiom set RAS from other user's ontologies. The formal definition of an IAS (RAS) has been given in Definition 32 in Section 4.3.

A MinCS is an IAS, a RAS, or union of both. As elaborated in Section 4.3, computing all IAS and RAS is tightly related to computing all explanations (also called

MinA) and diagnoses. The computation by a HST algorithm [Rei87] is repeated here only briefly. The HST algorithm makes repeated calls to an auxiliary procedure that computes one MinCS. A tree is built, where each node is labeled with a MinCS and each edge with an axiom. If the MinCS labeling a node has n axioms ($S := \{a_1, \ldots, a_n\}$), then this node is expanded with n children: the edge to the i-th child labeled with a_i, the child labeled with a MinCS that is not allowed to contain neither a_i nor any ancestor's edge label. This ensures that each node is labeled with a MinCS distinct from those of its predecessors.

HST optimizations such as *early termination* and *node reuse* avoid redundant computations and are included in current implementations. Another optimization is putting a *cardinality limit*, applicable when not all, but only the CS of minimal cardinality $|S|$ is of interest. Then nodes might contain partial solutions, called *partial MinCS*, in the sense that some axioms are not contained, but still a smallest CS is proven to be found (see Section 4.3).

Example 24. We continue Example 23. Assume we want to make c_1 as private as possible, i.e. $G = \{(c_1, \ell_0)\}$. All RAS are $\{a_1\}, \{a_2, a_3\}, \{a_4, a_5\}$, so the smallest MCS is $M_1 = \{(a_1, \ell_0)\}$ and we get $\mathsf{lbl}_{M_1}(c_1) = \ell_0$. As second example assume we want to make c_2 as public as possible, i.e. $G = \{(c_2, \ell_1)\}$. All IAS are $\{a_2\}, \{a_3\}$, so a smallest MCS is $M_2 = \{(a_3, \ell_1)\}$ and we get $\mathsf{lbl}_{M_2}(c_2) = \ell_1$.

Computing a Smallest MCS

A MCS for several goals is a set of several MinCS: one MinCS for each goal. However, it is no solution to compute single MinCS and combine them since this might not yield a smallest MCS or they might even conflict as the following example shows.

Example 25. We combine both goals of Example 24 simultaneously, i.e. we want to make c_1 as private as possible and c_2 as public as possible, $G = \{(c_1, \ell_0), (c_2, \ell_1)\}$. Just concatenating the above mentioned MCS to $M = M_1 \cup M_2 = \{(a_1, \ell_0), (a_3, \ell_1)\}$ is no MCS since $\mathsf{lbl}_M(c_2) = \ell_0 \neq \ell_1$. However, $M = \{(a_4, \ell_0), (a_5, \ell_0), (a_2, \ell_1)\}$ is a MCS.

For this reason we call any combination of MinCS a *candidate MCS (cMCS)*. To compute the smallest MCS, we introduce Algorithm 5.2 which is similar to the HST-based Algorithm 4.7 for computing the smallest CS in Section 4.3.2. The only difference is that each call to the auxiliary procedure computes a (partial) cMCS instead of a (partial) MinCS which is assigned to a node in the search tree, and edges are not labeled with an axiom but with a tuple (a, ℓ) which is not allowed in the child node's (partial) cMCS.

A (partial) cMCS is computed by a call $\mathsf{extract\text{-}partial\text{-}cMCS}(K, n)$ to the auxiliary procedure in Algorithm 5.1, where K is the set of prohibited label changes, i.e. all tuples at edges to ancestors in the HST, and n is the size of the currently known smallest MCS. The procedure includes 2 optimizations: *MinCS reuse* and *cardinality limit*. As any cMCS is a combination of MinCS, one MinCS might be contained in several cMCS. Instead of computing it anew for every cMCS, the first optimization reuses it. Putting a cardinality limit is a second optimization which computes a cMCS or stops once this has reached a size n and returns a potentially partial cMCS. As

Algorithm 5.1 Compute a (partial) cMCS, with optimizations MinCS reuse (disable by removing Lines 10, 11) and cardinality limit (disable by replacing in Line 7 "$n-|M|$" by "∞")

Procedure init-cMCS-extraction$(O, \mathsf{lab}, (L, \leqslant), G)$
Input: O, lab: labeled ontology; $(L, \leqslant)$: lattice; G: goal set

1: **Global:** $O, \mathsf{lab}, G' := \{(c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, C_S) \mid (c, \ell_g) \in G,$
$\mathrm{is}_I := \ell_g \not\leqslant \mathsf{lbl}(c) \wedge O_{\geqslant \ell_g} \not\models c,$ (decision to compute IAS)
$\mathrm{is}_R := \ell_g \not\geqslant \mathsf{lbl}(c) \wedge O_{\not\leqslant \ell_g} \models c,$ (decision to compute RAS)
$C_S := \emptyset\}$ (reuse set for MinCS)

Procedure extract-partial-cMCS(K, n)
Input: K: prohibited label changes; n: cardinality limit
Output: first n elements of a cMCS

1: $M := \emptyset$
2: **for** each goal $(c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, C_S) \in G'$ **do**
3: $H := \{a \mid (a, \ell_g) \in K\}$ (set of axioms not allowed to be labeled with ℓ_g)
4: **if** $\exists S' \in C_S : \emptyset = S' \cap H$ **then**
5: $S := S'$ (MinCS reuse)
6: **else**
7: $S :=$extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, H, n - |M|)$ (Algorithm 4.6)
8: **if** $\emptyset = S$ **then**
9: **return** $\emptyset$ (HST normal termination for one goal fires for complete goal set)
10: **if** $|S| \neq n - |M|$ **then**
11: $C_S := C_S \cup \{S\}$ (remember only non-partial MinCS)
12: $M := M \cup \{(a, \ell_g) \mid a \in S\}$
13: **return** M

will be discussed in the empirical evaluation of computing MinCS in Section 6.2.2, the cardinality limit optimization significantly reduces the time to find the smallest CS. In a partial cMCS, the last contained MinCS is always partial. Reuse is not done for partial MinCS.

Turning to Algorithm 5.2, whenever a cMCS M is found with $|M| < n$, it is smaller than our currently known smallest MCS and we can be sure that it is not partial. The question remains if it is a MCS or only a cMCS, which is checked in Line 6: neither is an axiom allowed to have multiple labels assigned (*syntactic conflict*) nor might a MinCS for one goal influence any other goal which is the case if any computed boundary does not equal the goal label (*semantic conflict).* Only after passing both checks, we update our globally known smallest known MCS $M_{\min}$ in Line 7. Loosening the constraints of a goal set, the semantic conflicts can be resolved in Line 10 or syntactic conflicts can be resolved in Line 12 which is explained in the next section.

We now show correctness of both optimizations, MinCS reuse and cardinality limit. Reuse of MinCS is correct, since the only non-constant parameter to extract a MinCS in Line 7 is the set of prohibited axioms H and Line 4 ensures H and the reused MinCS are disjoint.

Algorithm 5.2 Compute a smallest MCS by a HST algorithm

Procedure HST-extract-smallest-MCS$(O, \mathsf{lab}, (L, \leqslant), G, K)$
Input: O, lab: labeled ont.; $(L, \leqslant)$: lattice; G: goal set; K: prohibited label changes
Output: MCS of minimum cardinality

```
1: Global M_min := ∅, n := ∞, G
2: init-cMCS-extraction(O, lab, (L, ⩽), G)
3: expand-HST-MCS(K)
4: return M_min
```

Procedure expand-HST-MCS(K)
Input: K: prohibited label changes
Side effects: modifications to $M_{\min}$ and n

```
1: M := extract-partial-cMCS(K, n)
2: if M = ∅ then
3:    return                                   (HST normal termination)
4: if |M| < n then
5:    if (a, ℓ1), (a, ℓ2) ∈ M ⟹ ℓ1 = ℓ2 then
6:       if ∀(c, ℓg) ∈ G : lbl_M(c) = ℓg then
7:          M_min := M
8:          n := |M_min|
9:       else
10:         ...                               (semantic conflict resolution)
11:   else
12:      ...                                  (syntactic conflict resolution)
13: for the first (n − 1) label changes (a, ℓ) ∈ M do
14:    expand-HST-MCS(K ∪ {(a, ℓ)})
```

Theorem 5 (Cardinality Limit Optimization)**.** *Let O,*lab *be a labeled ontology and G a goal set. If m is the minimum cardinality of all MCS for G, the HST Algorithm 5.2 outputs a MCS M such that $|M| = m$.*

Proof. The described algorithm outputs the empty set or a MCS since the globally stored and finally returned $M_{\min}$ is only modified when the output of extract-partial-cMCS has size strictly smaller than the limit n, has neither any syntactic nor any semantic conflict and hence only when this is indeed a MCS itself. Suppose now that the output MCS $M_{\min}$ is such that $m < |M_{\min}|$, and let M_0 be a MCS such that $|M_0| = m$, which exists by assumption. Then, every MCS, i.e. every cMCS free of syntactic and semantic conflicts, obtained by calls to extract-partial-cMCS has size strictly greater than m, since otherwise, $M_{\min}$ and n would be updated. Consider now an arbitrary MCS M' found during the execution through a call to extract-partial-cMCS, and let $M'_n := \{(a_1, \ell_1), \ldots, (a_n, \ell_n)\}$ be the first n assignments of M'. Since M' is a (partial) MCS, it must be the case that $M_0 \not\subseteq M'_n$ since every returned MCS is minimal in the sense that no label change might be removed to obtain another MCS. Then, there must be an $i, 1 \leqslant i \leqslant n$ such that $(a_i, \ell_i) \notin M_0$. But then, M_0 will still be a MCS (and a cMCS anyway) after label change $\{(a_i, \ell_i)\}$ has been removed.

Since this argument is true for all nodes, it is in particular true for all leaf nodes, but then they should not be leaf nodes, since a new cMCS, namely M_0 can still be found by expanding the HST, which contradicts the fact that $M_{\min}$ is the output of the algorithm. □

5.2.4 Conflict Resolution

We already elaborated on syntactic and semantic conflicts which might prevent a cMCS from being a MCS. It might be the case that for a goal set, no MCS can be found.

Example 26. We continue Example 23. Assume $G = \{(c_1, \ell_4), (c_2, \ell_3)\}$. For the goal (c_1, ℓ_4) all IAS are $\{a_2\}, \{a_3\}$. For the goal (c_2, ℓ_3) all RAS are $\{a_1\}, \{a_2\}$. The cMCS $M_1 = \{(a_2, \ell_4), (a_2, \ell_3)\}$ is obviously no MCS due to a syntactic conflict. But also the remaining cMCS $M_2 = \{(a_2, \ell_4), (a_1, \ell_3)\}, M_3 = \{(a_3, \ell_4), (a_1, \ell_3)\}, M_4 = \{(a_3, \ell_4), (a_2, \ell_3)\}$ are no MCS due to semantic conflicts, since $\mathsf{lbl}_{M_2}(c_1) = \mathsf{lbl}_{M_3}(c_1) = \ell_3 \neq \ell_4$ and $\mathsf{lbl}_{M_4}(c_2) = \ell_4 \neq \ell_3$.

For these cases we introduce a generalization of a MCS called *relaxed MCS (RMCS)* where the goal set is only partially satisfied according to a defined strategy. For the special case of no conflict, the RMCS equals the MCS. We identified 4 strategies to resolve conflicts, where we focus on syntactic conflict resolution only:

1. Overrestrictive: accept lower labels for a minimal number of consequences than specified by the goal label. Formally, $\forall (c, \ell_g) \in G : \mathsf{lbl}_M(c) \neq \ell_g \implies \mathsf{lbl}_M(c) < \ell_g$ and cardinality $|\{(c, \ell_g) \in G \mid \mathsf{lbl}_M(c) \neq \ell_g\}|$ is minimal. Applied to the above example, $\{(a_2, \ell_3)\}$ is a RMCS.

2. Overpermissive: accept higher labels for a minimal number of consequences than specified by the goal label. Formally, $\forall (c, \ell_g) \in G : \mathsf{lbl}_M(c) \neq \ell_g \implies \mathsf{lbl}_M(c) > \ell_g$ and cardinality $|\{(c, \ell_g) \in G \mid \mathsf{lbl}_M(c) \neq \ell_g\}|$ is minimal. Applied to the above example, $\{(a_2, \ell_4)\}$ is a RMCS.

3. Override strategy: The goal G set is split up into fragments G_i so that $G = G_1 \cup \ldots \cup G_n$ for which individual MCS M_i can be computed. The changed label assignment $((\mathsf{lab}_{M_1}) \ldots)_{M_n}$ is obtained by sequentially applying each MCS M_i, where the order can be chosen based on some prioritization. This implies that labels changed by one MCS might be changed again by any subsequent MCS. Applied to the above example, splitting up G into G_1 and G_2, $G_1 = \{(c_1, \ell_4)\}$ yields MCS $M_5 = \{(a_2, \ell_4)\}$, subsequently $G_2 = \{(c_2, \ell_3)\}$ yields MCS $M_6 = \{(a_2, \ell_3)\}$.

Strategy 3 although easy to implement has an unacceptable drawback, conflicting our RMCS definition: even if there is a MCS for the union of all goal subsets, a sequentially applied MCS for one goal subset might override a previous for another goal subset since they are computed independently of each other. For this reason we focus on strategies 1 and 2 for resolution of syntactic conflicts.

Algorithm 5.3 describes the resolution of syntactic conflicts. It is an adapted version of Algorithm 5.2, where additionally the global variable r stores the minimal number

Algorithm 5.3 Compute a smallest RMCS by a HST algorithm, overpermissive strategy (for overrestrictive strategy, substitute in Line 3 "$\oplus$" for "$\otimes$", in Line 4 "$\geqslant$" for "$\leqslant$", and in Line 5 "$>$" for "$<$")

The following changes refer to the original Algorithm 5.2. In Procedure HST-extract-smallest-MCS, add global variables $N := \emptyset, r := \infty$, and add before Line 4:

```
1: if ∅ = M_min then
2:    return N
```

In Procedure expand-HST-MCS, replace Line 12 for syntactic conflict resolution with:

```
1: N' := M
2: for each a : (a, ℓ1), (a, ℓ2) ∈ N' ∧ ℓ1 ≠ ℓ2 do
3:    N' := N' \ {(a, ℓ1), (a, ℓ2)} ∪ {(a, ℓ1 ⊕ ℓ2)}
4: if ∀(c, ℓg) ∈ G : lbl_N'(c) ⩾ ℓg then          (fulfills overpermissive strategy)
5:    r' := |{(c, ℓg) ∈ G | lbl_N'(c) > ℓg}|
6:    if r' < r then
7:       N := N'
8:       r := r'
```

of overpermissive (overrestrictive) consequence labels and N stores the RMCS with minimal r. Again this Algorithm relies on the cMCS extraction Algorithm 5.1 and the optimization of reusing MinCS can be applied. The cardinality limit optimization is of no use here since if no MCS is found, then no cardinality limit is set and the HST is fully expanded.

There are goal sets yielding semantic conflicts but no syntactic conflicts in cMCS. These are not solvable by syntactic conflict resolution. For these cases not only IAS and RAS, but complete explanations and diagnoses need to be taken into account, as the following example shows.

Example 27. We continue Example 23. Assume $G = \{(c_1, \ell_2), (c_2, \ell_5)\}$. For the goal (c_1, ℓ_2) all IAS are $\{a_4\}, \{a_5\}$. For the goal (c_2, ℓ_5) all IAS are $\{a_2\}, \{a_3\}$, all RAS are $\{a_1\}, \{a_2, a_3\}$. Obviously no combination of MinCS for both goals yields a syntactic conflict. Nevertheless there is no MCS since every combination of MinCS has a semantic conflict. After semantic conflict resolution, an overpermissive RMCS is $N_{OP} = \{(a_4, \ell_2), (a_2, \ell_2 \oplus \ell_5 = \ell_1), (a_3, \ell_5\}$, yielding $\mathsf{lbl}_{N_{OP}}(c_1) = \ell_1, \mathsf{lbl}_{N_{OP}}(c_2) = \ell_1$. An overrestrictive RMCS is $N_{OR} = \{(a_4, \ell_2), (a_2, \ell_2 \otimes \ell_5 = \ell_0), (a_3, \ell_5)\}$, yielding $\mathsf{lbl}_{N_{OR}}(c_1) = \ell_5, \mathsf{lbl}_{N_{OR}}(c_2) = \ell_5$.

An empirical evaluation of our algorithms from this section is provided in Section 6.2.3.

5.3 Conclusions of the Chapter

The goal of this thesis is not only to provide a theoretically solid, but also a usable framework. Since security administrators might (yet) be unfamiliar with the task of assigning access restrictions to ontology axioms, we proposed two techniques.

The first proposal is to apply access restrictions that have been given to a document also to the respective document ontology and all contained axioms. In this case it might happen that the same axiom is contained in several document ontologies where the intuition would be that a user is allowed to read the axiom if she is allowed to read at least one of those documents.

The second proposal is to formulate an access restriction with a query. All responses to that query shall receive the respective restriction. We compared two basic approaches to enforce those query-based access restrictions: query rewriting vs. label filtering. Compared to query rewriting, label filtering allows higher knowledge availability in the sense that more answers are delivered to a user, while not uncovering any secret. Furthermore, it is independent of a concrete ontology language. However, it relies on a given axiom labeling. Starting from an arbitrary axiom labeling, e.g. a random labeling, the problem solved by our algorithms is to find a smallest MCS defining a new axiom labeling which enforces the query-based access restrictions. The query-based access restrictions are used to generate a so-called goal set, which consists of tuples with consequence and goal label. The access restrictions are enforced by the axiom labeling if the computed boundary for each consequence is equal to its goal label. This is a generalization of repairing one consequence's boundary, where changes on the axiom labeling for one consequence must not interfere with another consequence. We show that a changed label assignment does not always exist since a goal set might contain conflicts and we provide two conflict resolution strategies to relax the goal set, so that a relaxed changed label assignment can be computed.

Revisiting the research questions from Section 1.4, these results answer the questions 4, 5 and 6.

There are many examples of old, incorrect theories that stubbornly persisted, sustained only by the prestige of foolish but well-connected scientists. [...] Many of these theories have been killed off only when some decisive experiment exposed their incorrectness. [...] Thus the yeoman work in any science, and especially physics, is done by the experimentalist, who must keep the theoreticians honest.

Michio Kaku

6 Empirical Evaluation

On large real-world ontologies, we empirically evaluated implementations of the algorithms to

- compute a consequence's boundary from Section 4.2,
- repair a consequence's boundary from Section 4.3, and
- repair a set of consequence boundaries from Section 5.2.

We also empirically evaluated the availability gain of knowledge which can be delivered without telling secrets as discussed in Section 5.2 for query rewriting vs. label filtering. Since an empirical evaluation of results from Chapter 3 would not make sense, it is not contained here, but the results have been evaluated within the chapter already. All empirical results are presented collectively in this chapter, since large portions of the same test data and the test environment have been used throughout all tests. The following sections describe the test data and the test environment first and then present the empirical results which show that our algorithms perform well in practical scenarios.

6.1 Test Data and Test Environment

We performed our tests on a PC with 2GB RAM and Intel Core Duo CPU 3.16GHz. We implemented all approaches in Java 1.6 and for convenient OWL file format parsing and reasoner interaction we used the OWL API for OWL 2 [BVL03] in trunk revision 1150 from 21.5.2009.[1]

Labeling Lattices

Although we focus on comparing the efficiency of the presented algorithms, and not on practical applications of these algorithms, we have tried to use inputs that are

[1] Subversion Repository `https://owlapi.svn.sourceforge.net/svnroot/owlapi/owl1_1/trunk`

closely related to ones encountered in applications. The two labeling lattices $(L_d, \leqslant_d)$ and $(L_l, \leqslant_l)$ are similar to ones encountered in real-world applications. The labeling lattice $(L_d, \leqslant_d)$ was already introduced in Figure 1.3. Lattices of this structure (where the elements correspond to hierarchically organized user roles) can be obtained from a real-world access control matrix with the methodology presented in Section 3.2. The labeling lattice $(L_l, \leqslant_l)$ is a linear order with 6 elements $L_l = L_d = \{\ell_0, \ldots, \ell_5\}$ with $\leqslant_l := \{(\ell_n, \ell_{n+1}) \mid \ell_n, \ell_{n+1} \in L_l \wedge 0 \leqslant n \leqslant 5\}$, which could represent an order of trust values as in [Sch08] or dates from a revision history.

Ontologies, Label Assignment and Reasoners

We used the two ontologies O^{Snomed} and O^{Funct} with different expressivities and types of consequences for our experiments.

The Systematized Nomenclature of Medicine, Clinical Terms (Snomed ct) is a comprehensive medical and clinical ontology which is built using the DL $\mathcal{EL}^{++}$. Our version of O^{Snomed} is the January/2005 release of the DL version, which contains 379,691 concept names, 62 object property names, and 379,704 axioms. Since more than five million subsumptions are consequences of O^{Snomed}, testing all of them was not feasible and we used the same sample subset as described in [BS08], i.e., we sampled 0.5% of all concepts in each top-level category of O^{Snomed}. For each sampled concept A, all subsumptions $A \sqsubseteq B$ following from O^{Snomed} with A as subsumee were considered. Overall, this yielded 27,477 subsumptions. Following the ideas of [BS08], we precomputed the reachability-based module for each sampled concept A with the reasoner CEL 1.0 [BLS06] and stored these modules. This module is guaranteed to contain all axioms of any MinA and any diagnosis, thus also any IAS and RAS, for a subsumption $A \sqsubseteq B$ with A the considered subsumee. This module for A was then used as the start ontology when considering subsumptions with subsumee A, rather than searching the complete ontology.

The OWL ontology O^{Funct} has been designed for functional descriptions of mechanical engineering solutions and was presented in [GKL09; Gaa10]. It has 115 concept names, 47 object property names, 16 data property names, 545 individual names, 3,176 axioms, and the DL expressivity used in the ontology is $\mathcal{SHOIN}(\mathbf{D})$. Its 716 consequences are 12 subsumption and 704 instance relationships (concept assertions). To obtain labeled ontologies, axioms in both labeled ontologies received a random label assignment of elements from $L_l = L_d$. As black-box subsumption and instance reasoner we used the reasoner Pellet 2.0 [Sir+07], since it can deal with the expressivity of both ontologies. For the expressive DL $\mathcal{SHOIN}(\mathbf{D})$ it uses a tableau-based algorithm and for $\mathcal{EL}^{++}$ it uses an optimized classifier for the OWL 2 EL profile that is based on the algorithm described in [BBL05].

Test Setting for Access Restrictions to Implicit Knowledge

We tested access restrictions to implicit knowledge from Section 4.2 in the following setting. The boundary computation with full axiom pinpointing (FP) uses log-extract-MinA() (Algorithm 2 from [BS08], which is identical to Algorithm 8 from [Sun08]) and the HST based HST-extract-all-MinAs() procedure (Algorithm 9 from [Sun08]). The set

of extracted MinAs is then used to calculate the label of the consequence. We break after 10 found MinAs in order to limit the runtime, so there might be non-final label results. The boundary computation with label-optimized axiom pinpointing (LP) with min-lab() and HST-boundary() are implementations of Algorithm 4.1 and Algorithm 4.2 of Section 4.2.4. The boundary computation with binary search for linear ordering (BS in the following) implements Algorithm 4.3 of Section 4.2.4. We tested 8 combinations resulting from the 2 ontologies O^{Snomed} and O^{Funct}, 1 lattice $(L_d, \leqslant_d)$, 2 approaches FP and LP and the same 2 ontologies, 1 lattice $(L_l, \leqslant_l)$, and 2 approaches LP and BS.

Test Setting for Repairing Access Restrictions to Implicit Knowledge

We tested repairing access restrictions to implicit knowledge from Section 4.3 in the following setting. We took the computed boundary ℓ_c of each consequence c of the ontologies from the first experiment and then computed the MinCS to reach goal boundary ℓ_g which is constantly ℓ_3 in all experiments. Consequences were not considered if $\ell_c = \ell_g$. Thus, from the 716 consequences in O^{Funct}, we have 415 remaining with labeling lattice $(L_d, \leqslant_d)$ and 474 remaining with $(L_l, \leqslant_l)$. From the 27,477 consequences in O^{Snomed} we have 23,695 remaining with labeling lattice $(L_d, \leqslant_d)$ and 25,897 with $(L_l, \leqslant_l)$. The MinCS computation with FP uses the procedures log-extract-MinA() and the HST based HST-extract-all-MinAs(), implemented by the algorithms mentioned above. The MinCS computation with extract-partial-MinCS() and the smallest CS computation with HST-extract-smallest-CS() including optimizations for fixed axioms and cardinality limit are implementations of Algorithm 4.6 and Algorithm 4.7 of Section 4.3.2. The required IAS and RAS extraction with extract-partial-IAS(), extract-partial-RAS() are implementations of Algorithms 4.4 and 4.5 also from Section 4.3.2. We break after 10 found MinAs (or respectively MinCS or partial MinCS) in order to limit the runtime, so there might be no computed MinCS at all or a non-smallest MinCS returned. We tested 12 combinations resulting from the 2 ontologies O^{Snomed} and O^{Funct}, 2 labeling lattices $(L_d, \leqslant_d)$ and $(L_l, \leqslant_l)$ and 3 variants (FP, fixed axioms, and fixed axioms in combination with cardinality limit). Running the fixed axioms optimization without cardinality limit can be done easily by skipping Line 11 in Algorithm 4.7.

Test Setting for Query-Based Access Restrictions

We tested query-based access restrictions to implicit knowledge from Section 5.2 in the following setting. The labeling lattice is $(L_d, \leqslant_d)$ from which we use the top lattice element ℓ_1 for public knowledge, ℓ_2 for intermediate knowledge and ℓ_3 for top secret knowledge. The set of test ontologies is extended with ontologies containing a large ABox, including O^{Geom},[2] O^{MGED},[3] O^{Process}.[4] They are selected ontologies from the TONES Ontology Repository[5] with a high number of individuals. At time of their

[2] http://i2geo.net/ontologies/dev/ontology.owl
[3] http://mged.sourceforge.net/ontologies/MGEDOntology.owl
[4] http://sweet.jpl.nasa.gov/ontology/process.owl
[5] http://owl.cs.manchester.ac.uk/repository/

download on 25.3.2010, they had the characteristics given in Table 6.1. The ontology O^{FUNCT}, which has been introduced above already, is also included.

In a first test setting we evaluated the availability of knowledge in the presence of access control by query rewriting vs. access control by label filtering. Initially each ontology axiom is labeled ℓ_1 (public). This reflects a default allow-all behavior of a security policy. Then for each concept C in the ontology, we apply an access restriction defined by a query $AR = C(x)$ and add every query result $c_i = \mu(AR)$ with goal label ℓ_3 (top secret) to the goal set. A smallest MCS is computed for this goal set. The computed MCS is used to create a newly labeled ontology, on which we perform the following queries. We count for every C-instance the instance relationships to concepts other than C which are available for public users (ℓ_1). With query rewriting their count is 0, due to the principle of query rewriting explained in Section 5.2.2. With label filtering their count might be greater than 0 and is considered to be the availability gain of label filtering vs. query rewriting. For cMCS extraction defined by Algorithm 5.1, we tested both optimizations MinCS reuse and cardinality limit separately and in their combination. In this setting every cMCS is automatically a MCS since there are no conflicting goals. Although not included in Algorithm 5.2 for transparency reasons, the mentioned usual HST optimizations *early termination* and *node reuse* are included in our implementation.

In a second test setting we evaluated conflict resolution strategies in cases where multiple goals conflict each other, so that no MCS can be computed without relaxing one of the goals. We test the overrestrictive conflict resolution approach vs. the overpermissive conflict resolution approach of Algorithm 5.3 with the same extended set of ontologies used for the first test setting. From the two optimizations available for the auxiliary procedure to extract a cMCS in Algorithm 5.1, only MinCS reuse but not cardinality limit is applied. The latter is useless in the presence of conflicting goals for reasons explained in Section 5.2.4. First all axioms are labeled with intermediate security level, i.e. ℓ_2. A goal set is created for each concept C containing the same consequences described above, but now one arbitrarily chosen half of this set has goal label ℓ_1 and the other half has goal label ℓ_3. This has the effect that some of the resulting goal sets are contradictory. We test Algorithm 5.3 to compute a RMCS with overpermissive vs. overrestrictive conflict resolution strategy for the same goal set and we count the number of overpermissive (respectively overrestrictive) consequence labels.

For both settings, the test data characteristics are given in Table 6.1. The number of goal sets and of goals per goal set are the same for both experiments, since they contain the same concept assertions, but with different goal labels. In order to limit runtime, we compute in maximum 10 cMCS before the HST Algorithm 5.2 (respectively HST Algorithm 5.3) returns, so there might be a smaller MCS we do not find (respectively we might find no RMCS although there is one, or there might be a RMCS we do not find which has fewer overpermissive respectively overrestrictive consequence labels).

Ontology	DL expressivity	♯logical axioms	♯concepts	♯individuals	♯goal sets	♯goals per goal set
O^{Funct}	$\mathcal{ALCOIN}(\mathbf{D})$	3189	115	545	102	12.2
O^{Geom}	$\mathcal{ALCHOIN}(\mathbf{D})$	8803	589	2010	571	14.1
O^{Process}	$\mathcal{ALCHOF}(\mathbf{D})$	2578	1537	150	40	20.9
O^{MGED}	$\mathcal{ALEOF}(\mathbf{D})$	1387	234	681	125	28.8

Table 6.1: Test sets consisting of ontologies and goal sets

6.2 Experimental Results

Our experiments show that our algorithms perform well with practical large-scale ontologies. In the following we describe our empirical results of each of the following tasks with labeled ontologies:

- computing a boundary to discover access restrictions to a given consequence,
- for a single consequence, repair the boundary by changing axiom labels, and
- for a set of consequences, repair the boundaries by changing axiom labels.

6.2.1 Access Restrictions to Implicit Knowledge

The results for boundary computation by FP vs. LP, using lattice $(L_d, \leqslant_d)$ and the two ontologies O^{Snomed} and O^{Funct} are given in Table 6.2. The table consists of an upper part and a lower part. The upper part contains a set of consequences that is "easy," because each consequence has fewer than 10 MinAs. This contains 21,001 subsumptions from O^{Snomed} and 307 consequences from O^{Funct}. The lower part contains a set of consequences that is "hard," because each consequence has at least 10 MinAs. This contains 6,476 subsumptions from O^{Snomed} and 409 consequences from O^{Funct}.

While LP computed the boundary for each consequence following from the easy and the hard set, FP computed the boundary for each consequence following from the easy but not for each following from the hard set. As described above, we break after 10 found MinAs. A label computed for a consequence following from the hard set, called non-final label, might be lower than the boundary since there might be further MinAs providing a higher label. For a practical system, a lower label puts an unnecessary strong access restriction to a consequence, resulting in an overrestrictive policy.

For the easy set of O^{Snomed}, the overall labeling time for all 21,001 subsumptions with FP was 50.25 minutes. For LP it was 1.50 minutes, which means that LP is about 34 times faster than FP. For the hard set of O^{Snomed}, the non-final labels of FP were identical to the boundaries of LP in 6,376 of the 6,476 cases (98%), i.e., in most cases the missing MinAs would not have changed the already computed label. FP took 2.5 hours without final results, whereas LP took 0.6% (factor 155) of that time and returned final results after 58 seconds. We started a test series limiting runs of FP

			♯early termination	♯reuse	♯calls to extract MinA (MinLab)	♯MinA (♯MinLab)	♯axioms (♯labels) per MinA (MinLab)	Lattice operations time in ms	Total labeling time in ms
O^{Snomed} easy	FP	avg	81.05	9.06	26.43	2.07	5.40	0.25	143.55
		max	57,188.00	4,850.00	4,567.00	9.00	28.67	45.00	101,616.00
		stddev	874.34	82.00	90.48	1.86	3.80	0.86	1,754.03
	LP	avg	0.01	0.00	2.76	1.03	1.73	0.35	4.29
		max	2.00	1.00	6.00	3.00	3.00	57.00	70.00
		stddev	0.13	0.02	0.59	0.16	0.56	0.98	3.62
O^{Funct} easy	FP	avg	43.59	29.52	26.56	4.26	3.05	0.49	3,403.56
		max	567.00	433.00	126.00	9.00	6.50	41.00	13,431.00
		stddev	92.16	64.04	30.90	2.84	1.01	2.38	3,254.25
	LP	avg	0.09	0.02	2.80	1.33	1.40	0.76	207.32
		max	2.00	1.00	7.00	4.00	3.00	22.00	1,295.00
		stddev	0.34	0.13	0.90	0.54	0.48	1.56	87.29
O^{Snomed} hard	FP	avg	432.11	42.25	126.54	10.20	16.38	0.30	1,378.66
		max	42,963.00	5,003.00	4,623.00	16.00	37.80	14.00	148,119.00
		stddev	1,125.06	121.15	186.33	0.49	5.00	0.54	3,493.02
	LP	avg	0.04	0.00	3.12	1.06	2.05	0.32	8.88
		max	3.00	2.00	6.00	3.00	3.00	46.00	86.00
		stddev	0.21	0.04	0.50	0.25	0.44	1.04	4.26
O^{Funct} hard	FP	avg	30.01	16.00	26.44	10.04	4.41	0.56	8,214.91
		max	760.00	511.00	411.00	11.00	6.50	3.00	25,148.00
		stddev	85.33	47.79	40.61	0.20	1.08	0.55	3,428.97
	LP	avg	0.09	0.01	2.76	1.38	1.32	0.77	200.55
		max	3.00	2.00	7.00	4.00	2.00	16.00	596.00
		stddev	0.33	0.12	0.91	0.64	0.43	1.40	61.11

Table 6.2: Boundary computation by FP vs. LP, using lattice $(L_d, \leqslant_d)$ and two ontologies with an easy and a hard set of consequences

to <30MinAs, which did not terminate after 90 hours, with 1,572 labels successfully computed and 30 subsumptions skipped since they had ⩾30MinAs. Interestingly, in both the easy and the hard set, LP can rarely take advantage of the optimizations early termination and reuse, which might be due to the simple structure of the lattice.

Similar results have been observed for the easy and the hard set of O^{Funct}. Again, the computation of FP was restricted to <10 MinAs. This time, only 363 out of 409 (88%) non-final labels of FP were equal to the boundaries of LP. Although the ontology is quite small, LP again performs much better than FP. The reason could be that in this ontology, consequences frequently have a large set of MinAs.

For a system designer, a question to decide could be to use either (a) our approach of keeping one large ontology with labeled axioms and precompute all consequences[6] and their labels or (b) the naïve approach of managing separate ontologies and computing

[6] When we say "all consequences" we always mean "all consequences the user is interested in." These might be, e.g., all concept assertions of named individuals to named concepts and all subsumptions between two named concepts. This restriction is necessary since already from simple axioms, infinitely

all consequences of each separate ontology independently. We can make the following rough estimate while we assume that the lattice is nonlinear but the details of its structure would not have any influence. Based on our test results with O^{SNOMED}, an estimate for the time needed to compute a label for all the more than 5 million subsumptions in O^{SNOMED} with LP would be $2.47 \cdot \frac{5 \cdot 10^6}{27477} \approx 449$ minutes. Assuming 20 minutes to compute all consequences with the current CEL reasoner [Sun08], our approach to compute and label all consequences would be as expensive as computing all consequences $\frac{20+449}{20} \approx 23$ times. Two remarks have to be made here.

1. Taking the fast computation time of 20 minutes, achieved by CEL is to some extent unfair, since the slower (see [Sun08] for a comparison) Pellet is used in our experiments for reasons explained above. However, at the time of these experiments, Pellet fails to classify the complete O^{SNOMED} because of memory exhaustion. For this reason we process only the reachability-based modules with Pellet and not the complete O^{SNOMED}, as described above. Presumably, the realistic ratio is actually below 23.

2. The preparation step computing the reachability-based modules as described above took < 0.21 seconds [BS08] and can be neglected here.

Our approach is as expensive as computing 23 views if we assume that computing all consequences for each of the views requires 20 minutes. For incomparable user labels, e.g. representing user roles which do not inherit permissions from each other while one user can have several roles, already the considerably low number of 5 incomparable user labels implies $2^5 = 32$ views and our approach is faster. For fewer user labels, the naïve approach is faster. Based on our test results with O^{FUNCT}, a similar estimate can be made. Computing all consequences requires 5 seconds and labeling all consequences with LP requires 146 seconds. In this case our approach is as expensive as computing all consequences 30 times. Again with 5 or more user labels, our approach is faster.

Apart from this numerical estimate, there are further reasons contra the several ontologies approach and pro our labeling approach. Maintaining one ontology is very error-prone, and maintaining several simultaneously is even more. If an axiom is changed, it should be changed in all of them. Hence, even if labeling all consequences takes longer, this is paid-off by the certainty that the changes have been propagated to all ontologies, i.e. less changes are necessary by the ontology engineer and less tests must be done to check the ontology. There is also a reduction on space used, but since space is inexpensive nowadays one may want to ignore this.

In statistics, a histogram is often used to roughly assess the probability distribution. The range of values on the x-axis is divided into non-overlapping intervals and the y-axis provides the number of observations. The histogram in Figure 6.1 shows, for all 4 combinations of the 2 ontologies and the 2 computation methods, the number of MinAs (respectively MinLabs) required to compute a boundary or non-final label. From this histogram and also from Table 6.2, one can see that LP requires at most

many nonequivalent consequences may follow, e.g. from $M \sqsubseteq \exists l.M$ it follows that $M \sqsubseteq \exists l.M, M \sqsubseteq \exists l.(\exists l.M)$, etc.

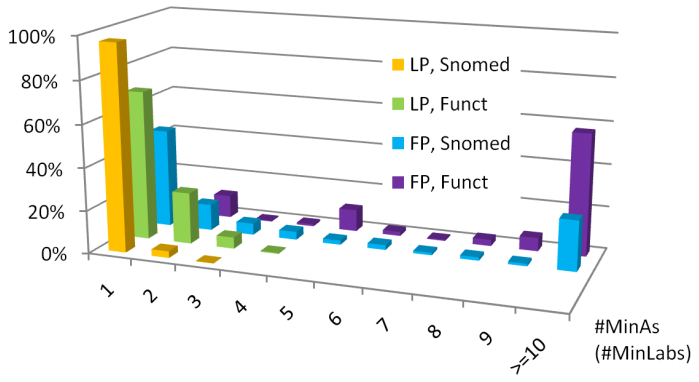

Figure 6.1: Histogram of required ♯MinAs (♯MinLabs) to compute a boundary (respectively non-final label)

three MinLabs for O^{Snomed}, at most four for O^{Funct}, and usually just one MinLab whereas FP usually requires more MinAs.

The histograms in Figure 6.2 compare the distribution of time needed with FP vs. LP to compute a boundary of a consequence from the union of the above described easy and hard set. The required time is given on the logarithmic x-axis, where the number below each interval defines the maximum contained value. As can be seen, FP takes more time than LP in general. Note that moving to the left on the x-axis means a relatively high performance improvement, due to the logarithmic scale. It can be further seen that LP covers a few intervals while FP covers more. This indicates that FP has a higher variability and the standard deviation values in Table 6.2 confirm this.

Table 6.3 provides results for LP vs. BS with the total order $(L_l, \leqslant_l)$ as labeling lattice. For O^{Snomed}, LP takes 130.4 and BS takes 77.1 seconds to label all 27,477 subsumptions. For O^{Funct}, LP takes 133.9 and BS takes 68.6 seconds to label all 716 consequences. Interestingly, labeling all consequences of O^{Funct} or all consequences of O^{Snomed} takes roughly the same time, perhaps due to a trade-off between ontology size and expressivity. Roughly, BS is twice as fast (factor 1.7 with O^{Snomed}, 1.9 with O^{Funct}) compared to LP.

Above, we already discussed the decision of a system designer whether to use our approach or the naïve approach for nonlinear lattices. Based on our test results a similar estimate can be made for linear lattices. Labeling all consequences of O^{Snomed} would require $\frac{77.1}{60} \cdot \frac{5 \cdot 10^6}{27477} \approx 234$ minutes. Similar to the explanation above, our approach is as expensive as computing all consequences $\frac{20+234}{20} \approx 13$ times, i.e. with 13 or more user labels our approach is faster. For O^{Funct}, our approach is as expensive as computing all consequences $\frac{68.8}{5} \approx 14$ times, i.e. with 14 or more user labels our approach is faster.

The histograms in Figure 6.3 compare the distribution of time needed to compute a boundary with BS vs. LP. Again the required time is given on the logarithmic x-axis. They show that the performance gain with BS over LP is higher with O^{Funct}

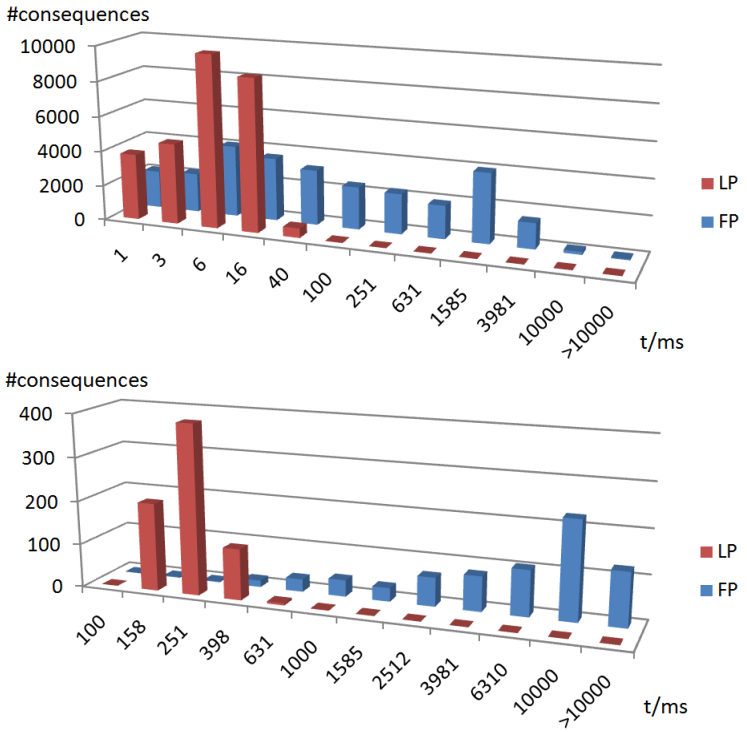

Figure 6.2: Histograms of time needed to compute a consequence's boundary in O^{Snomed} (upper) and O^{Funct} (lower) with the methods FP vs. LP

compared to O^{Snomed}, as discussed already. They further show that there is no clear winner with respect to variability, as it has been the case comparing FP vs. LP and Table 6.3 confirms that observation.

		LP							BS	
		♯early termination	♯reuse	♯calls to extract MinLab	♯MinLab	♯labels per MinLab	Lattice oper. time	Total labeling time	Iterations	Total labeling time
O^{SNOMED}	avg	0.03	0.00	2.24	1.03	1.23	0.37	4.75	2.41	2.81
	max	1.00	0.00	5.00	3.00	2.00	329.00	330.00	3.00	75.00
	stddev	0.18	0.00	0.45	0.19	0.42	4.85	6.37	0.49	2.94
O^{FUNCT}	avg	0.09	0.00	2.50	1.27	1.24	0.82	186.98	2.55	95.80
	max	1.00	0.00	5.00	3.00	2.00	62.00	1147.00	3.00	877.00
	stddev	0.28	0.00	0.72	0.49	0.40	2.74	69.55	0.50	45.44

Table 6.3: Boundary computation by LP vs. BS on a sampled set of 27,477 subsumptions in O^{SNOMED}/ all 716 consequences of O^{FUNCT} with lattice $(L_l, \leqslant_l)$ (time in ms)

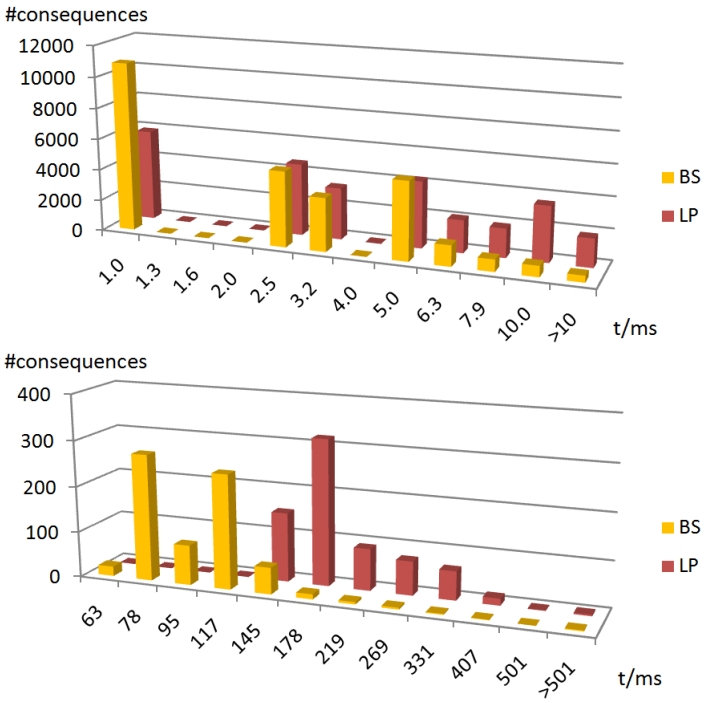

Figure 6.3: Histograms of time needed to compute a consequence's boundary in O^{SNOMED} (upper) and O^{FUNCT} (lower) with the methods BS vs. LP

Ont.	Lattice	Variant	Runtime limit per goal	Time in minutes	Ratio of correct solutions	Ratio of optimal solutions
O^{Funct}	$(L_d, \leqslant_d)$	FP	$\leqslant 10$ MinA	44.05	96%	47%
		fixed axioms	$\leqslant 10$ MinCS	17.56	100%	90%
		fixed axioms, card. lim.	$\leqslant 10$ (partial) MinCS	8.65	100%	98%
	$(L_l, \leqslant_l)$	FP	$\leqslant 10$ MinA	54.46	98%	49%
		fixed axioms	$\leqslant 10$ MinCS	15.97	100%	96%
		fixed axioms, card. lim.	$\leqslant 10$ (partial) MinCS	8.61	100%	99%
O^{Snomed}	$(L_d, \leqslant_d)$	FP	$\leqslant 10$ MinA	184.76	100%	75%
		fixed axioms	$\leqslant 10$ MinCS	15.87	100%	99%
		fixed axioms, card. lim.	$\leqslant 10$ (partial) MinCS	10.51	100%	100%
	$(L_l, \leqslant_l)$	FP	$\leqslant 10$ MinAs	185.35	100%	75%
		fixed axioms	$\leqslant 10$ MinCS	40.83	100%	95%
		fixed axioms, card. lim.	$\leqslant 10$ (partial) MinCS	28.14	100%	98%

Table 6.4: Results comparing variants to compute a smallest CS

6.2.2 Repairing Access Restrictions to Implicit Knowledge

Table 6.4 contains results for the 4 combinations of the 2 ontologies and the 2 labeling lattices. For each of them we tested 3 variants, leading to 12 test series overall. As described above, we limit the number of computed MinAs and MinCS to 10, so our algorithms might not find any, or not a smallest change set before reaching the limit. We measure the quality of the presented variants given this limitation at execution time in the following sense. Table 6.4 lists the ratio of correct solutions where at least 1 correct MinCS was computed, and the ratio of optimal solutions where the limit was not reached during the computation and thus yielded the smallest change set possible. Notice however, that the ratio of cases with the smallest change set successfully computed might be higher, including those where the limitation was reached but the smallest change set was already found.

Figure 6.4 depicts a time-quality diagram of all variants from Table 6.4, where quality is the ratio of correct multiplied by the ratio of optimal solutions. Obviously, a desirable variant is in the upper left corner yielding maximum quality in minimal time. It can be seen that FP is clearly outperformed by our optimizations. The experiment shows that fixed axioms and cardinality limit, especially in their combination, are optimizations yielding significantly higher quality and lower runtime.

Instead of providing histograms of time needed to repair a consequence's boundary, we provide the cumulative distribution in Figure 6.5. The difference to the histograms is that not discrete intervals, but instead the continuous spectrum of time needed is depicted, and the number of consequences is cumulated over time until it reaches the number of all considered consequences. For this reason, the maximum value on the y-axis, from left to right and top to bottom, is: 415, 474, 23695 and 25897. The

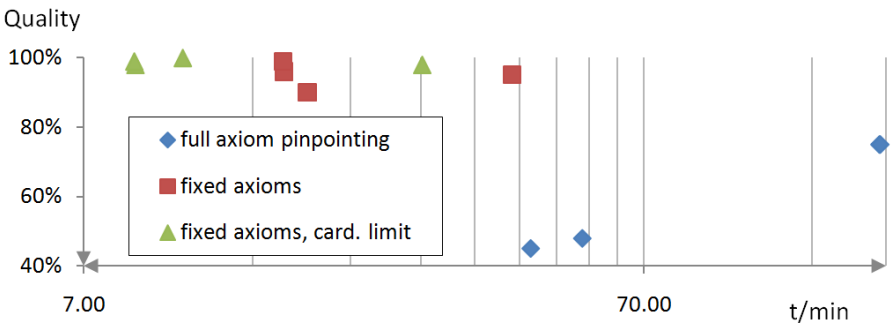

Figure 6.4: Time-quality diagram comparing variants to compute a smallest CS

reason for those numbers of consequences has been explained above. The x-axis is again logarithmic, as it has been the case with the histograms above. It can be seen that in general a single consequence from O^{SNOMED} is repaired much faster than one from O^{FUNCT}. Interestingly, even at the start of the logarithmic x-axis, some consequences are repaired already. In our test result log, the reported time for several consequences was even 0 ms. The reason is Java's limitation with respect to measuring fractions of milliseconds. It can be seen, as already known from Table 6.4, that for the two diagrams of O^{SNOMED} and the upper right diagram of O^{FUNCT}, most of the consequences are repaired roughly one order of magnitude faster with both of our optimizations enabled compared to the naïve FP approach.

6.2.3 Query-Based Access Restrictions

The experimental results for the first experiment on query-based access restrictions are given in Table 6.5. It compares availability of access control by query rewriting vs. access control by label filtering and it compares performance of the optimizations cardinality limit vs. MinCS reuse. The given total number of MinCS includes reused MinCS. The number of cMCS is equal to the number of MCS, since the goals contain no conflicts with the first experiment. The number of gained assertions confirms that our ideas improve availability of knowledge when using label filtering instead of query rewriting. While the number of gained assertions is comparable between the optimizations applied, their runtime differs significantly. MinCS reuse alone, and also in combination with cardinality limit runs significantly faster compared to using cardinality limit optimization only. Testing O^{MGED} with cardinality limit optimization did not terminate after 4 days, so no results are provided.

The diagram in Figure 6.6 plots gained assertions over the number of goals contained in a goal set. For all four ontologies, the number of gained assertions increases frequently with an increasing goal set size. This is because no conflicts are contained in the goal sets in this first experiment. The diagrams show test results with both optimizations enabled, since we explained above that the number of gained assertions does not depend on the optimizations.

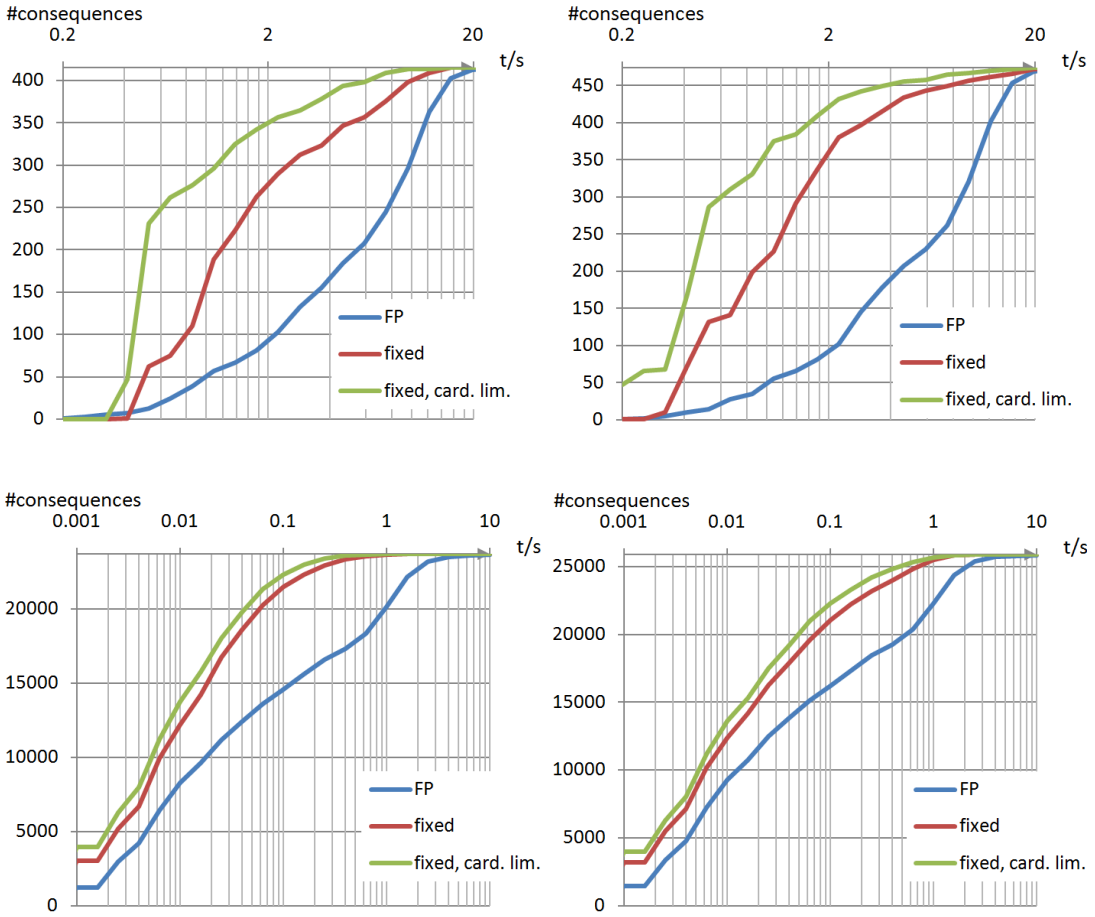

Figure 6.5: Cumulative distribution of time needed to repair a consequence's boundary in O^{Funct} (upper) and O^{Snomed} (lower) with the lattices $(L_d, \leqslant_d)$ (left) and $(L_l, \leqslant_l)$ (right)

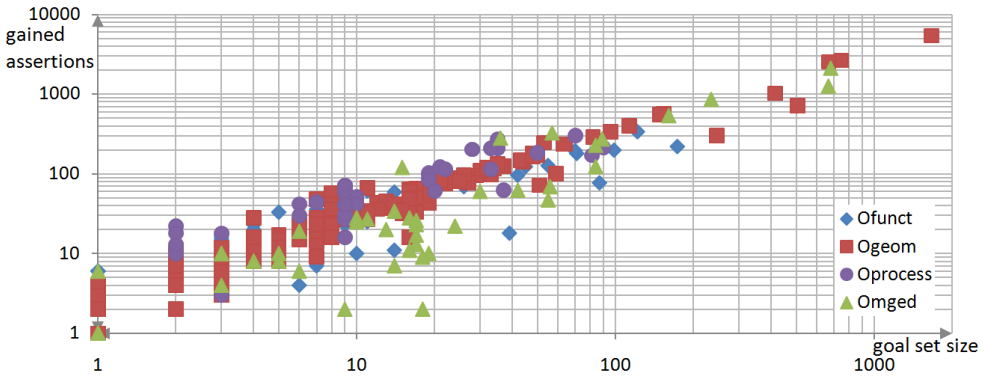

Figure 6.6: Gained assertions over goal set size

Test set	Optimization	Results (averages per goal set)					
		♯MinCS	♯reused MinCS	♯cMCS = ♯MCS	|MCS|	Time in minutes	♯gained assertions
O^{Funct}	card. limit	131.8	0.0	3.9	23.9	3.6	28.5
	MinCS reuse	135.2	118.4	3.9	24.0	0.7	28.6
	both	132.6	115.7	3.9	24.1	0.6	28.4
O^{Geom}	card. limit	146.9	0.0	2.6	9.2	24.0	43.4
	MinCS reuse	148.9	132.9	2.5	9.3	4.2	43.3
	both	147.3	131.1	2.6	9.3	4.2	43.3
O^{Process}	card. limit	199.3	0.0	6.9	12.0	2.3	92.6
	MinCS reuse	250.9	217.8	6.7	12.2	0.6	91.8
	both	197.9	165.0	6.8	12.2	0.6	91.9
O^{MGED}	card. limit	n/a	n/a	n/a	n/a	n/a	n/a
	MinCS reuse	286.4	253.4	2.9	15.1	115.9	53.9
	both	265.1	232.4	3.0	15.1	114.3	54.1

Table 6.5: Comparison of gained assertions compared to query rewriting and performance of optimizations

The diagrams in Figure 6.7 plot the required time to compute a smallest MCS over the number of goals per goal set for all four ontologies. As expected, the required time increases with the number of goals in a goal set. The diagrams and Table 6.5 show that the optimization of putting a cardinality limit is significantly slower than reusing MinCS while both optimizations combined are close to using the latter optimization only.

The experimental results for the second experiment comparing conflict resolution with overrestrictive strategy vs. overpermissive strategy are given in Table 6.6. Only some of the goal sets constructed as described above are conflicting, and results are only given for those. Only the given percentage of the goals of a goal set are enforced, the remaining consequences have overpermissive (respectively overrestrictive) labels making them more public (private) than intended by the goal set. The runtime limit of 10 cMCS was hit in every case, making the HST algorithm stop with non-final results. In our case it always computed at least one RMCS, but as explained above there might be a RMCS with fewer overpermissive (overrestrictive) consequence labels when relaxing this runtime limit. The number of RMCS equals the number of cMCS in all our tests, so there is no conflict which cannot be solved by syntactic conflict resolution in our test set, although these cases can be constructed as explained in Section 5.2.4.

The number of overpermissive (respectively overrestrictive) consequences is plotted against the number of goals in a goal set in the upper part of Figure 6.8. It shows that the more conflicting goals are contained in a goal set, the more consequence labels do not receive the originally intended label but an overrestrictive (overpermissive) label

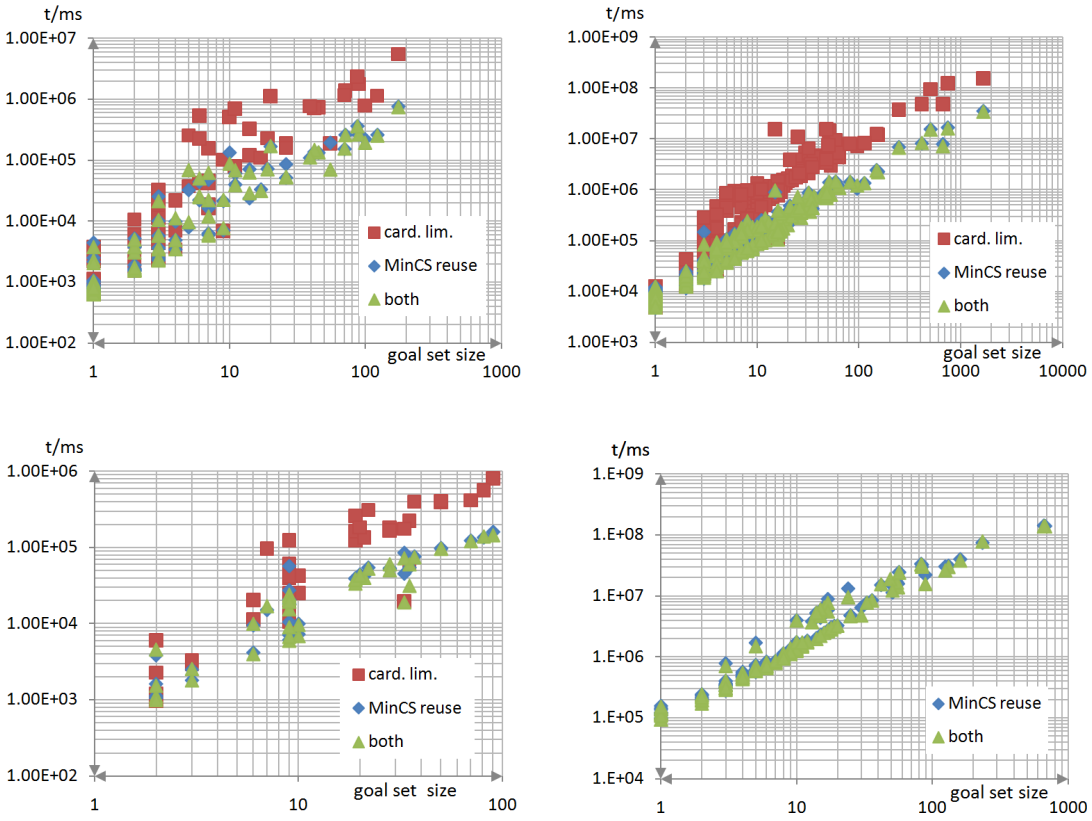

Figure 6.7: Required time to compute a smallest MCS over the number of goals per goal set with ontologies O^{Funct} (upper left), O^{Geom} (upper right), O^{Process} (lower left), O^{MGED} (lower right)

assigned. The lower part of Figure 6.8 depicts the percentage of enforced goals plotted against the number of goals in a goal set. In our tests set there does not seem to be a relation between the two, so a larger set of conflicting goals does not necessarily mean that a lower percentage of those goals are enforced.

6.3 Conclusions of the Chapter

We evaluated implementations of our algorithms empirically, showing that they perform well in practical scenarios with large-scale ontologies. Our experiments cover the algorithms for computing a consequence's boundary, repairing this boundary and repairing the boundaries for several consequences simultaneously:

- Computing a consequence's boundary is one of the reasoning tasks with labeled ontologies. From the two black-box algorithms that can deal with arbitrary lattices, the full axiom pinpointing approach is clearly outperformed by the label-optimized axiom pinpointing approach, which is faster up to factor 155. For the

Test set	♯goal sets confl.	♯goals per confl. goal set	Strat-egy	Results (averages per conflicting goal set) ♯cMCS	♯RMCS	|RMCS|	Time in minutes	♯OR/OP cons. labels	% of enforced goals
O^{Funct}	19	50.3	OR	10.0	10.0	101.4	2.2	19.5	64%
			OP	10.0	10.0	110.0	2.0	20.3	64%
O^{Geom}	39	150.7	OR	10.0	10.0	139.4	45.4	63.3	71%
			OP	10.0	10.0	140.4	37.0	52.1	80%
O^{Process}	23	31.0	OR	10.0	10.0	32.3	0.9	12.7	62%
			OP	10.0	10.0	32.6	0.8	11.0	67%
O^{MGED}	16	165.8	OR	10.0	10.0	140.4	814.6	75.6	59%
			OP	10.0	10.0	141.6	780.8	51.9	73%

Table 6.6: Conflict resolution with overrestrictive (OR) strategy vs. overpermissive (OP) strategy

special case where the labeling lattice is a total order, label-optimized axiom pinpointing is again outperformed by the Binary Search approach by roughly factor 2. We provided an estimate from a system designer's point of view comparing our approach of labeled ontologies and consequences to a naïve approach of reasoning over separate ontologies. It showed that our approach is faster when more than four incomparable user labels are present in a nonlinear lattice or when more than 12 user labels are present in a linear lattice.

- A second reasoning task is repairing a consequence's boundary, which is only possible by changing axiom labels. Our experiments show that our algorithms and optimizations yield tangible improvements in both the execution time and the quality of the proposed smallest CS defining a new axiom labeling. In order to compute a smallest CS, the approach of computing all MinAs is outperformed up to factor 12 by our optimized approach of computing IAS and RAS. Limiting cardinality further reduces computation time up to factor 2. In combination we observed a performance increase up to factor 18. But not only performance is improved, at the same time both optimizations increase quality of the computed smallest CS under limited resources at runtime.

- A third reasoning task is repairing several consequence boundaries simultaneously. A comparison of our label filtering approach to a query rewriting approach revealed that label filtering allows making more knowledge available while still keeping the defined secrets. For example, for an ontology with 150 individuals and 1537 concepts, we gained on average 90 concept assertions per goal set. We found that, the larger a conflict-free goal set, the more assertions are gained in general, but the required time also increases. To reduce time, especially our optimization of MinCS reuse pays off and increases performance by almost factor 6 compared to the cardinality limit optimization. In the presence of conflicts in a

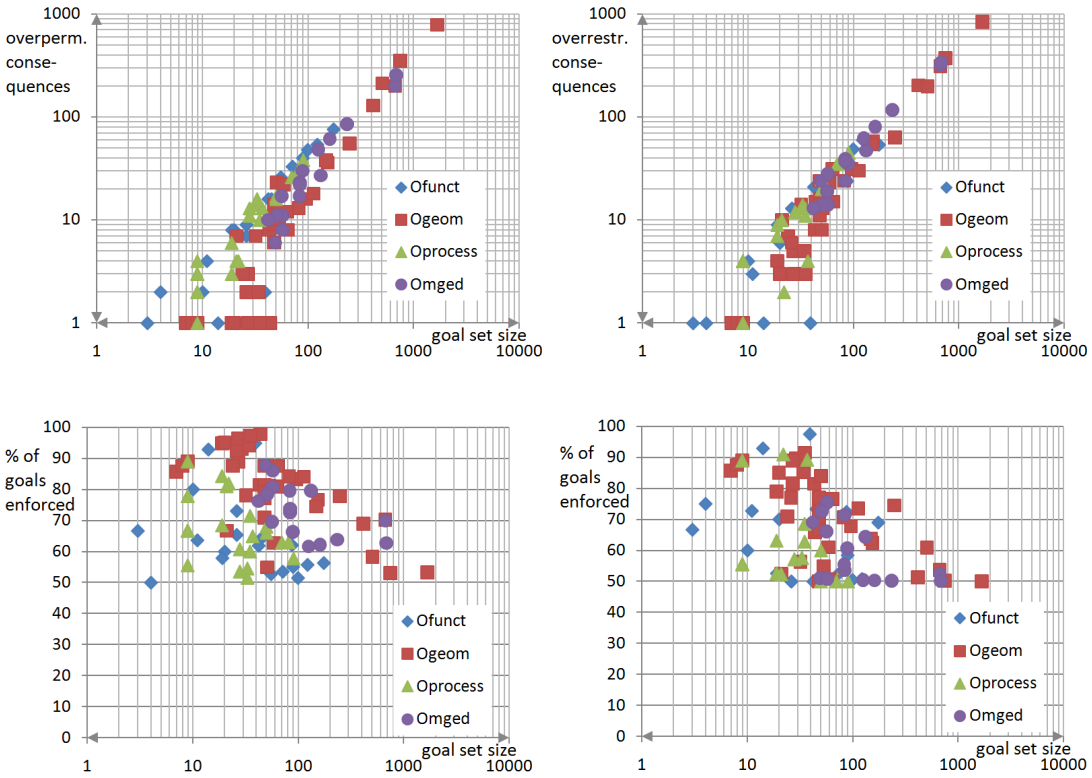

Figure 6.8: Overpermissive (left) and overrestrictive (right) conflict resolution with the number of overriden consequence labels (upper) and percentage of enforced goals (lower) over the number of goals in a goal set

goal set, an overpermissive or overrestrictive conflict resolution still allows enforcing 59%-80% of the goals contained in a goal set in our scenario. Interestingly, this percentage did not generally decrease with larger goal sets.

In general, for all experiments we observed longer computation times for more expressive ontologies. For this reason, in practical applications it might help to keep the used expressivity as low as absolutely necessary.

Vision without execution is hallucination.

Thomas Edison

7 Conclusions

This thesis investigated several techniques to enable access restrictions *to* as well as *with* Description Logic ontologies. Central objectives of this work have been the representation and completion of RBAC policies and a framework for handling restrictions for reading access to an ontology's axioms and its consequences and how to handle them in a practically feasible and intuitive way. In the following sections, major technical and empirical results of this thesis are discussed and some directions for future work are given.

7.1 Discussion of Achieved Results

The major results achieved in the context of this thesis can be categorized into four groups:

- a methodology to represent RBAC policies with Description Logics and to complete RBAC policies systematically during a computer-supported dialog with the security engineer;
- a framework, which is independent of a specific ontology language or reasoner, for lattice-based access restrictions for reading access to explicit (and implicit) knowledge represented in an ontology (and following from that ontology);
- user support to facilitate the work of a security engineer to assign axiom-level access restrictions;
- an empirical evaluation, showing that concepts and algorithms developed in this thesis perform well in practical scenarios with large-scale ontologies.

All research questions from Section 1.4 have been answered as detailed out in the following. The following sections are a collection of the conclusions of the chapters.

7.1.1 Representing and Completing RBAC Policies

RBAC is a standardized and well established access control model, abstracting from individual users by user roles. We have shown how to represent RBAC policies as DL ontology and how to complete RBAC policies in the sense of adding new knowledge that has been neither explicit nor implicit before.

In Section 1.4 we posed the following research questions, which have been answered with results of this thesis.

7. Are DLs an appropriate formal representation of the explicit permissions defined by an RBAC policy and is it possible to infer the implicit permissions from this representation?
8. Can a labeling lattice be obtained from an existing RBAC matrix and what may need to be changed?
9. Can the systematic and non-redundant identification of constraints in an RBAC matrix be reduced to a known problem and what extensions may be necessary?

The main argument to use a knowledge representation formalism to represent an RBAC policy is that a policy specifies explicit permissions from which implicit permissions follow. We discussed approaches from literature to represent RBAC policies by means of a DL ontology and focused specifically on an extension of RBAC, called RBAC-CH, adding a hierarchy of object classes to RBAC. We introduced a representation of RBAC-CH that has several advantages over the proposal in related work. Furthermore, we have shown that the RBAC matrix can be computed from an RBAC policy by just plugging in a reasoner and querying for all implicit permissions. The resulting matrix helps, e.g., the security engineer to verify the effect of user role and object class hierarchy. However, we pointed out that a DL representation does not add any new knowledge that was not already contained in the RBAC policy. Basically, in addition to RBAC policy and RBAC matrix, it just adds a third representation based on Description Logics.

We have shown how to complete RBAC policies in the sense of adding new knowledge that has not yet been represented in a computer, but was known to an expert, e.g. the security engineer, before. Our methodology is based on a known method from FCA, called attribute exploration. An RBAC matrix can be written as three-dimensional formal context, but needs to be interpreted and transformed appropriately, so that attribute exploration for two-dimensional contexts can be applied. In one of the discussed interpretations, the strict interpretation, the set of permissions an individual can have is strictly defined by the RBAC matrix. No additional constraints need to be checked. Under a permissive (respectively prohibitive) interpretation, individuals have additional (respectively fewer) permissions than given for their user roles. This means the RBAC matrix is no complete definition of permissions for an individual. In that case we might nevertheless have general rules and constraints which have to be fulfilled in any case on the level of individual users. For example, nobody might be

allowed to write and approve the same document for a proper review process. Rules like this on the level of individual users and individual documents can be identified by our systematic and non-redundant method based on attribute exploration. We have shown that known implications from a user role hierarchy or object class hierarchy can be exploited in order to pose fewer questions to the security engineer. We introduced a new DL representation to capture knowledge from the RBAC matrix as well as the newly obtained knowledge.

Our results on the representation and completion of RBAC policies laid the ground for the next chapters on access restrictions to an ontology's axioms, since some of the results allow to reuse a given RBAC policy to prepare a labeling lattice which is required in our framework for access restrictions to an ontology's axioms and consequences.

7.1.2 Access Restrictions to Explicit and Implicit Knowledge

In Section 1.4 we posed the following research questions, which have been answered with results of this thesis.

1. For an access control model which allows multiple visible sub-ontologies from one large ontology with respect to a criterion expressed by axiom labels, user labels and a hierarchy between those labels, how can such a model be formalized and what constraints need to be taken into account?
2. Given that all axioms of an ontology are labeled, what is the label of a consequence which can be inferred and made explicit by a reasoner?
3. Can the repair of a consequence label be reduced to a known problem, and what optimizations are possible?

We have considered a scenario where ontology axioms are labeled and user labels determine views on the ontology, i.e. sub-ontologies that are obtained by comparing the user label with the axiom labels. The labeled axioms entail consequences, but their labels are not explicitly given. However, intuitively a consequence should have the same access restriction as the axioms from which it follows. We introduced an approach formalizing this intuition and showed how to compute a label for a given consequence, called boundary. Our approach can be used for large-scale ontologies since, on the one hand, it allows to pre-compute consequences without having to do this separately for all possible views. Once we have computed a boundary for the consequence, checking whether this consequence follows from a sub-ontology is reduced to a simple label comparison. On the other hand, the fact that we employ a black-box approach for computing the boundary allows us to use existing highly-optimized reasoners, rather than having to implement a new reasoner from scratch.

Our general framework allows for using any restriction criterion that can be represented using a lattice, such as user roles, levels of trust, granularity, or degrees of certainty. With user roles, each axiom label defines the user roles able to see the axiom and each user label defines the sub-ontology containing the axioms visible to this user.

In the presence of trust restrictions, the user label specifies the trust level required for the ontology axiom. This supports scenarios with axioms from different sources, like company-internal with high trust level and public Web with low trust level. In the presence of uncertainty, e.g. in possibilistic reasoning, each axiom has an associated certainty degree in the interval $[0, 1]$. The user label then specifies the certainty degree required for the axioms and the consequences. Similarly, granularity restrictions (i.e., on how much details the ontology should provide for the user) can be expressed by a total order.

Our framework is independent of a specific reasoner. To stay as general as possible, we do not fix a specific ontology language. We just assume that ontologies are finite sets of axioms such that every subset of an ontology is again an ontology and we require the consequence relation to be monotonic. The elements of the labeling lattice which are used as user label have to be join-prime.

The security administrator might not be satisfied with the access restriction level computed for a consequence from the access restriction levels of the axioms that entail the consequence. In this case, the computed boundary is different from the intended label and a repair of the consequence's boundary is required. This is only indirectly possible by changing some axiom labels. Based on ontology repair techniques we developed algorithms to compute a change set of minimal cardinality, which contains axioms to be relabeled in order to yield a desired boundary for a given consequence. The base problem, finding a smallest MinA and diagnosis without computing all of them allows for applications beyond our domain. Our algorithms take advantage of (1) fixing a subset of the axioms which are known not to be part of the search space and (2) limiting cardinality of MinCS to be computed in the Hitting Set Tree to the size of the smallest known change set. All our algorithms are black-box based, which means that they can be used with any off-the-shelf reasoner, without the need for modifications.

7.1.3 User Support to Assign Access Restrictions to Ontology Axioms

The goal of this thesis is not only to provide a theoretically solid, but also a usable framework. Since security administrators might (yet) be unfamiliar with the task of assigning access restrictions to ontology axioms, we proposed two techniques.

In Section 1.4 we posed the following research questions, which have been answered with results of this thesis.

4. How can repair of a consequence label be extended to the repair of multiple consequence labels in parallel?
5. How are conflicts handled that might appear when goal labels of multiple consequences interfere with each other?
6. Is it possible to increase usability by reusing document access restrictions for document ontologies or by assigning a label to a set of consequences intentionally defined by a query?

The first proposal is to apply access restrictions that have been given to a document also to the respective document ontology and to all contained axioms. In this case it might happen that the same axiom is contained in several document ontologies where the intuition would be that a user is allowed to read the axiom if she is allowed to read at least one of those documents.

The second proposal is to formulate an access restriction with a query. All responses to that query shall receive the respective restriction. We compared two basic approaches to enforce those query-based access restrictions: query rewriting vs. label filtering. Compared to query rewriting, label filtering allows higher knowledge availability in the sense that more answers are delivered to a user, while not uncovering any secret. Furthermore, it is independent of a concrete ontology language. However, it relies on a given axiom labeling. Starting from an arbitrary axiom labeling, e.g. a random labeling, the problem solved by our algorithms is to find a smallest MCS defining a new axiom labeling which enforces the query-based access restrictions. The query-based access restrictions are used to generate a so-called goal set, which consists of tuples with consequence and goal label. The access restrictions are enforced by the axiom labeling if the computed boundary for each consequence is equal to its goal label. This is a generalization of repairing one consequence's boundary, where changes on the axiom labeling for one consequence must not interfere with another consequence. We show that a changed label assignment does not always exist since a goal set might contain conflicts and we provide two conflict resolution strategies to relax the goal set, so that a relaxed changed label assignment can be computed.

7.1.4 Empirical Evaluation

We evaluated implementations of our algorithms empirically, showing that they perform well in practical scenarios with large-scale ontologies. Our experiments cover the algorithms for computing a consequence's boundary, repairing this boundary and repairing the boundaries for several consequences simultaneously:

- Computing a consequence's boundary is one of the reasoning tasks with labeled ontologies. From the two black-box algorithms that can deal with arbitrary lattices, the full axiom pinpointing approach is clearly outperformed by the label-optimized axiom pinpointing approach, which is faster up to factor 155. For the special case where the labeling lattice is a total order, label-optimized axiom pinpointing is again outperformed by the Binary Search approach by roughly factor 2. We provided an estimate from a system designer's point of view comparing our approach of labeled ontologies and consequences to a naïve approach of reasoning over separate ontologies. It showed that our approach is faster when more than four incomparable user labels are present in a nonlinear lattice or when more than 12 user labels are present in a linear lattice.

- A second reasoning task is repairing a consequence's boundary, which is only possible by changing axiom labels. Our experiments show that our algorithms and optimizations yield tangible improvements in both the execution time and the quality of the proposed smallest CS defining a new axiom labeling. In order

to compute a smallest CS, the approach of computing all MinAs is outperformed up to factor 12 by our optimized approach of computing IAS and RAS. Limiting cardinality further reduces computation time up to factor 2. In combination we observed a performance increase up to factor 18. But not only performance is improved, at the same time both optimizations increase quality of the computed smallest CS under limited resources at runtime.

- A third reasoning task is repairing several consequence boundaries simultaneously. A comparison of our label filtering approach to a query rewriting approach revealed that label filtering allows making more knowledge available while still keeping the defined secrets. For example, for an ontology with 150 individuals and 1537 concepts, we gained on average 90 concept assertions per goal set. We found that, the larger a conflict-free goal set, the more assertions are gained in general, but the required time also increases. To reduce time, especially our optimization of MinCS reuse provides advantages by increasing performance by almost factor 6 compared to the cardinality limit optimization. In the presence of conflicts in a goal set, an overpermissive or overrestrictive conflict resolution still allows enforcing 59%-80% of the goals contained in a goal set in our scenario. Interestingly, this percentage did not generally decrease with larger goal sets.

In general, for all experiments we observed longer computation times for more expressive ontologies. For this reason, in practical applications it might help to keep the used expressivity as low as absolutely necessary.

7.2 Directions for Future Work

Our framework might be extended and improved by research in several directions, in order of the respective chapters:

- For the representation and completion of RBAC policies, a follow up might be to seek for a smaller DL fragment which meets the modeling requirements for RBAC policies. This is particularly essential for the permissive approach, as the DL modeling we used so far is based on some non-standard DL constructors. Next, one could want to support positive and negative authorizations in one policy, i.e. combine the permissive and prohibitive approach. Finally, recall that our approach was based on the assumption that the sets of roles and document types are fixed. At changes of the sets, the obtained RBAC matrix needs to be considered as a new matrix and our procedure would have to be started from scratch. In some applications this might be too strict. The three interpretations would have to be adapted and even attribute exploration for the strict approach might make sense when dropping this assumption.

- The results on restricting access to ontology axioms and consequences are limited to reading access. The approach might be extended to other operations on axioms such as writing, deleting or changing.

- The influence of characteristics of practical labeling lattices could be investigated. A first estimate showed that our approach of labeled ontologies and consequences pays off compared to a naïve approach of reasoning over separate ontologies when more than four incomparable user labels are present in a nonlinear lattice or when more than 12 user labels are present in a linear lattice. However, further conclusions might be interesting on the influence of more specific lattice and user label characteristics.

- For labeled ontologies, one important reasoning task was to repair a consequence's boundary. As proposed, the computed smallest change set is minimal with respect to the number of changed axiom label assignments. Alternative criteria could be the number of changed consequence boundaries, the distance of the new from the old label in the lattice, or the number of users receiving changed permissions. Another direction might be to provide a more flexible goal definition language. Currently, a goal is defined to be equal to a selected label. A useful extension might be to define the goal to be greater than or equal, lower than or equal, or in other relation to a given label.

- A generalization of repairing a consequence's boundary was repairing a set of consequence boundaries, defined by a goal set. While we discussed syntactic conflict resolution for contradicting goal sets, one might also investigate the resolution of semantic conflicts. Furthermore, one could make the selection of the conflict resolution strategy more fine-grained in order to define for each single goal of a goal set whether it may be lowered or raised in case of conflicts.

- The performance of label filtering vs. query rewriting might be compared. While label filtering turned out to be superior to query rewriting with respect to the availability of knowledge, the empirical evaluation unveiled that repairing a set of boundaries is time consuming. On the one hand, this task can be performed offline before the system is running and label filtering would allow quick response times due to the computationally cheap label comparison. For this reason it might be especially useful in applications where low response times are important. On the other hand, in applications where the overall computation time is important, query rewriting might be more suitable. This might contain applications with a low number of queries until the ontology changes where the expensively computed labeling would be used for a short time only. This trade-off might be compared to the trade-off whether or not to compute an index structure for efficient access to a dataset. However, this hypothesis needs to be verified.

- New results for problems and improved implementations of algorithms on which this thesis is based could enable additional optimizations and could further increase performance in practical scenarios. This might include decision procedures, reasoning techniques, and pinpointing techniques. Examples might be the following. Distributed reasoning might enable a higher performance in a networked computing environment. Known explanations for some consequences

might be exploited for "similar" consequences based on a criterion to be identified. Incremental reasoning might turn out to be interesting especially for linear (parts of) labeling lattices.

Although the above open research directions have been opened up by results of this thesis, we believe that the results presented in this thesis already will facilitate adoption of and increase interest in Description Logic ontologies in enterprise environments and any other environments that require putting access restrictions to ontology axioms as well as the consequences following from that ontology axioms.

Bibliography

[And01] R. Anderson. *Security Engineering*. Available at `http://www.cl.cam.ac.uk/~rja14/book.html`. Wiley, 2001. ISBN: 0-471-38922-6. (Cited on page 35).

[And08] R. Anderson. *Security Engineering: A Guide to Buildung Dependable Distributed Systems*. 2nd edition. Selected chapters available at `http://www.cl.cam.ac.uk/~rja14/book.html`. Wiley, 2008. ISBN: 978-0-470-06852-6. (Cited on pages 32, 34, 35).

[Baa+07a] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. "Completing Description Logic Knowledge Bases using Formal Concept Analysis." In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*. AAAI Press, 2007. (Cited on page 44).

[Baa+07b] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. 2nd edition. Cambridge University Press, 2007. ISBN: 978-0521876254. (Cited on pages 2, 17, 18, 20, 23, 24).

[BBL05] F. Baader, S. Brandt, and C. Lutz. "Pushing the $\mathcal{EL}$ Envelope." In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*. Edinburgh, UK: Morgan-Kaufmann Publishers, 2005. (Cited on pages 24, 104).

[Bec+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference*. W3C Recommendation. Available at `http://www.w3.org/TR/owl-ref/`. World Wide Web Consortium, Feb. 2004. (Cited on pages 19, 48).

[BEL08] J. Biskup, D. W. Embley, and J.-H. Lochner. "Reducing inference control to access control for normalized database schemas." In: *Information Processing Letters* 106.1 (2008), pages 8–12. (Cited on page 76).

[Bis09] J. Biskup. *Security in Computing Systems: Challenges, Approaches, and Solutions*. Springer, 2009. ISBN: 9783540784418. (Cited on pages 35, 36, 62).

[BKP09a] F. Baader, M. Knechtel, and R. Peñaloza. *Computing Boundaries for Reasoning in Sub-Ontologies*. LTCS-Report 09-02. Available at `http://lat.inf.tu-dresden.de/research/reports.html`. Germany: Chair of Automata Theory, Institute of Theoretical Computer Science, Technische Universität Dresden, 2009. (Cited on page 61).

[BKP09b] F. Baader, M. Knechtel, and R. Peñaloza. "A Generic Approach for Large-Scale Ontological Reasoning in the Presence of Access Restrictions to the Ontology's Axioms." In: *Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*. Edited by A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan. Volume 5823. Lecture Notes in Computer Science. 2009, pages 49–64. (Cited on pages 61, 85).

[BL85] R. J. Brachman and H. J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985. ISBN: 978-0934613019. (Cited on pages 133, 136, 137).

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. "The Semantic Web." In: *Scientific American* 284 (May 2001), pages 29–37. (Cited on page 19).

[BLS06] F. Baader, C. Lutz, and B. Suntisrivaraporn. "CEL—A Polynomial-time Reasoner for Life Science Ontologies." In: *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR 2006)*. Edited by U. Furbach and N. Shankar. Volume 4130. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2006, pages 287–291. (Cited on pages 19, 104).

[BM08] N. Boustia and A. Mokhtari. "Representation and Reasoning on ORBAC: Description Logic with Defaults and Exceptions Approach." In: *Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES 2008)*. IEEE Computer Society, 2008. (Cited on page 38).

[BP08] F. Baader and R. Peñaloza. "Automata-Based Axiom Pinpointing." In: *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR 2008)*. Edited by A. Armando, P. Baumgartner, and G. Dowek. Lecture Notes in Artificial Intelligence 5195. Springer, 2008. (Cited on pages 25, 63, 68).

[BP10a] F. Baader and R. Peñaloza. "Automata-based Axiom Pinpointing." In: *Journal of Automated Reasoning* 45.2 (Aug. 2010). Special Issue: Selected Papers from IJCAR 2008, pages 91–129. (Cited on page 77).

[BP10b] F. Baader and R. Peñaloza. "Axiom Pinpointing in General Tableaux." In: *Journal of Logic and Computation* 20.1 (2010). Special Issue: Tableaux and Analytic Proof Methods, pages 5–34. (Cited on pages 25, 63, 68, 77, 95).

[BPS07] F. Baader, R. Peñaloza, and B. Suntisrivaraporn. "Pinpointing in the Description Logic $\mathcal{EL}$." In: *Proceedings of the 30th German Conference on Artificial Intelligence (KI 2007)*. Volume 4667. Lecture Notes in Artificial Intelligence. Osnabrück, Germany: Springer-Verlag, 2007, pages 52–67. (Cited on pages 25, 68, 69, 78).

[Bra83] R. J. Brachman. "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks." In: *IEEE Computer* 16.10 (1983), pages 30–36. ISSN: 0018-9162. (Cited on page 18).

[Bru+07] J.-S. Brunner, L. Ma, C. Wang, L. Zhang, D. C. Wolfson, Y. Pan, and K. Srinivas. "Explorations in the use of Semantic Web technologies for product information management." In: *Proceedings of the 16th international conference on World Wide Web (WWW 2007)*. (Banff, Alberta, Canada). New York, NY, USA: ACM, 2007, pages 747–756. ISBN: 978-1-59593-654-7. (Cited on page 2).

[BS08] F. Baader and B. Suntisrivaraporn. "Debugging SNOMED CT Using Axiom Pinpointing in the Description Logic $\mathcal{EL}^+$." In: *Proceedings of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED 2008)*. Phoenix, Arizona, 2008. (Cited on pages 25, 68, 81, 104, 109).

[BSH07] J. Bao, G. Slutzki, and V. Honavar. "Privacy-Preserving Reasoning on the Semantic Web." In: *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI 2007)*. Washington, DC, USA: IEEE Computer Society, 2007, pages 791–797. ISBN: 0-7695-3026-5. (Cited on pages 35, 93).

[Bur96] P. Burmeister. *Formal Concept analysis with ConImp: Introduction to Basic Features*. Technical report. TU Darmstadt, Germany, 1996. (Cited on page 29).

[BVL03] S. Bechhofer, R. Volz, and P. W. Lord. "Cooking the Semantic Web with the OWL API." In: *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*. Edited by D. Fensel, K. P. Sycara, and J. Mylopoulos. Volume 2870. Lecture Notes in Computer Science. 2003, pages 659–675. (Cited on page 103).

[Cal+08] D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. "View-Based Query Answering over Description Logic Ontologies." In: *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*. 2008. (Cited on page 93).

[CCT07] L. Cirio, I. F. Cruz, and R. Tamassia. "A Role and Attribute Based Access Control System Using Semantic Web Technologies." In: *Proceedings of On the Move to Meaningful Internet Systems Workshop (OTM 2007)*. 2007. (Cited on page 38).

[Coh04] M. Cohn. *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004. ISBN: 978-0321205681. (Cited on page 6).

[CS07a] J.-H. Chae and N. Shiri. "Description Logic Framework for Access Control and Security in Object-Oriented Systems." In: *Proceedings of the 11th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC 2007)*. 2007, pages 565–573. (Cited on page 38).

[CS07b] J.-H. Chae and N. Shiri. "Formalization of RBAC Policy with Object Class Hierarchy." In: *Proceedings of Information Security Practice and Experience (ISPEC)*. Edited by E. Dawson and D. S. Wong. Volume 4464. Lecture Notes in Computer Science. Springer, 2007, pages 162–176. ISBN: 978-3-540-72159-8. (Cited on pages 38, 39, 40, 41, 42, 43, 44).

[CS09] W. Chen and H. Stuckenschmidt. "A Model-driven Approach to enable Access Control for Ontologies." In: *Tagungsband 9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*. Edited by H. R. Hansen, D. Karagiannis, and H.-G. Fill. 2009, pages 663–672. (Cited on pages 35, 92, 93).

[Den76] D. E. Denning. "A lattice model of secure information flow." In: *Communications of the ACM* 19.5 (1976), pages 236–243. ISSN: 0001-0782. (Cited on pages 32, 33).

[Den82] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982. ISBN: 0-201-10150-5. (Cited on page 35).

[DK09] F. Dau and M. Knechtel. "Access Policy Design Supported by FCA Methods." In: *Proceedings of the 17th International Conference on Conceptual Structures (ICCS 2009)*. Edited by F. Dau and S. Rudolph. Volume 5662. Lecture Notes in Computer Science. 2009, pages 141–154. (Cited on page 38).

[DK10] F. Dau and M. Knechtel. "Systems and Methods for Generating Constraints for use in Access Control." Patent (United States). Application Number 12/823,884. Date of application 25.6. 2010. (Cited on page 38).

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. 2nd edition. Cambridge University Press, 2002. ISBN: 9780521784511. (Cited on pages 26, 28).

[Fin+08] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. "ROWLBAC: representing role based access control in OWL." In: *Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT 2008)*. (Estes Park, CO, USA). New York, NY, USA: ACM, 2008, pages 73–82. ISBN: 978-1-60558-129-3. (Cited on page 38).

[FJ02] C. Farkas and S. Jajodia. "The inference problem: a survey." In: *ACM SIGKDD Explorations Newsletter* 4.2 (2002), pages 6–11. ISSN: 1931-0145. (Cited on pages 35, 76).

[Flo+09] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. "Coloring RDF Triples to Capture Provenance." In: *Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*. Edited by A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan. Volume 5823. Lecture Notes in Computer Science. Heidelberg: Springer, 2009, pages 196–212. ISBN: 978-3-642-04929-3. (Cited on page 63).

[Gaa10] A. Gaag. "Entwicklung einer Ontologie zur funktionsorientierten Lösungssuche in der Produktentwicklung." PhD thesis. Munich University of Technology, 2010. (Cited on page 104).

[Gan84] B. Ganter. *Two Basic Algorithms in Concept Analysis*. Technical report Preprint-Nr. 831. Technische Hochschule Darmstadt, 1984. (Cited on page 30).

[GH08] B. C. Grau and I. Horrocks. "Privacy-preserving Query Answering in Logic-based Information Systems." In: *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*. 2008. (Cited on page 93).

[GKL09] A. Gaag, A. Kohn, and U. Lindemann. "Function-based solution retrieval and semantic search in mechanical engineering." In: *Proceedings of the 17th International Conference on Engineering Design (ICED 2009)*. 2009. (Cited on page 104).

[GKT07] T. J. Green, G. Karvounarakis, and V. Tannen. "Provenance semirings." In: *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS 2007)*. (Beijing, China). New York, NY, USA: ACM, 2007, pages 31–40. ISBN: 978-1-59593-685-1. (Cited on pages 63, 64).

[GO04] B. Ganter and S. A. Obiedkov. "Implications in Triadic Formal Contexts." In: *Proceedings of the 12th International Conference on Conceptual Structures (ICCS 2004)*. Edited by K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach. Volume 3127. Lecture Notes in Computer Science. Springer, 2004, pages 186–195. ISBN: 3-540-22392-4. (Cited on page 52).

[Gru93] T. R. Gruber. "A translation approach to portable ontology specifications." In: *Knowledge Acquisition* 5.2 (1993), pages 199–220. ISSN: 1042-8143. (Cited on page 21).

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1999. ISBN: 3-540-62771-5 284. (Cited on page 31).

[Hay79] P. J. Hayes. "The Logic of Frames." In: *Frame Conceptions and Text Understanding*. Edited by D. Metzing. Republished in [BL85]. Walter de Gruyter and Co., 1979, pages 46–61. (Cited on page 18).

[HBPK08] M. Heinrich, A. Boehm-Peters, and M. Knechtel. “MoDDo - a tailored documentation system for model-driven software development.” In: *Proceedings of the IADIS International Conference WWW/Internet (ICWI 2008)*. Edited by P. Isaías, M. B. Nunes, and D. Ifenthaler. 2008, pages 321–324. (Cited on page 4).

[HBPK09] M. Heinrich, A. Boehm-Peters, and M. Knechtel. “A Platform to Automatically Generate and Incorporate Documents into an Ontology-Based Content Repository.” In: *Proceedings of the 2009 ACM Symposium on Document Engineering (DocEng 2009)*. Edited by U. M. Borghoff and B. Chidlovskii. 2009, pages 43–46. (Cited on page 4).

[Hei+06] N. Heilili, Y. Chen, C. Zhao, Z. Luo, and Z. Lin. “An OWL-Based Approach for RBAC with Negative Authorization.” In: *Proceedings of the 1st International Conference on Knowledge Science, Engineering and Management (KSEM 2006)*. Edited by J. Lang, F. Lin, and J. Wang. Volume 4092. Lecture Notes in Computer Science. Springer, 2006, pages 164–175. ISBN: 3-540-37033-1. (Cited on page 38).

[HKS06] I. Horrocks, O. Kutz, and U. Sattler. “The Even More Irresistible SROIQ.” In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*. 2006. (Cited on pages 23, 41).

[HLP07] F. van Harmelen, V. Lifschitz, and B. Porter, editors. *Handbook of Knowledge Representation (Foundations of Artificial Intelligence)*. Elsevier Science, Dec. 2007. ISBN: 978-0-444-52211-5. (Cited on page 17).

[HM01] V. Haarslev and R. Möller. “RACER System Description.” In: *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR 2001)*. 2001, pages 701–706. (Cited on page 19).

[HMS04] U. Hustadt, B. Motik, and U. Sattler. “Reducing SHIQ-Description Logic to Disjunctive Datalog Programs.” In: *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004)*. 2004, pages 152–162. (Cited on page 19).

[Hor02] I. Horrocks. “DAML+OIL: a Reason-able Web Ontology Language.” In: *Proceedings of 8th Conference on Extending Database Technology*. Volume 2287. Lecture Notes in Computer Science. Springer-Verlag, 2002, pages 2–13. (Cited on page 19).

[IO05] C. M. Ionita and S. L. Osborn. “Specifying an Access Control Model for Ontologies for the Semantic Web.” In: *Proceedings of Secure Data Management, 2nd VLDB Workshop*. 2005. Chapter Specifying an Access Control Model for Ontologies for the Semantic Web, pages 73–85. (Cited on page 91).

[JF06] A. Jain and C. Farkas. "Secure resource description framework: an access control model." In: *Proceedings of the 11th ACM symposium on Access control models and technologies (SACMAT 2006)*. (Lake Tahoe, California, USA). New York, NY, USA: ACM, 2006, pages 121–129. ISBN: 1-59593-353-0. (Cited on page 63).

[Kal+05] A. Kalyanpur, B. Parsia, E. Sirin, and J. A. Hendler. "Debugging unsatisfiable classes in OWL ontologies." In: *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 3.4 (2005), pages 268–293. (Cited on pages 25, 63, 77).

[Kal+07] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. "Finding All Justifications of OWL DL Entailments." In: *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC+ASWC 2007)*. Volume 4825. Lecture Notes in Computer Science. Busan, Korea: Springer-Verlag, 2007, pages 267–280. (Cited on pages 25, 64, 68, 69, 72, 80, 85, 95).

[KH08] M. Knechtel and J. Hladik. "RBAC Authorization Decision with DL Reasoning." In: *Proceedings of the IADIS International Conference WWW/Internet (ICWI 2008)*. Edited by P. Isaías, M. B. Nunes, and D. Ifenthaler. 2008, pages 169–176. (Cited on pages 38, 39).

[KHD08] M. Knechtel, J. Hladik, and F. Dau. "Using OWL DL Reasoning to decide about authorization in RBAC." In: *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*. Edited by C. Dolbear, A. Ruttenberg, and U. Sattler. Volume 432. CEUR Workshop Proceedings. 2008. (Cited on pages 38, 39).

[KHP07] V. Kolovski, J. Hendler, and B. Parsia. "Analyzing Web access control policies." In: *Proceedings of the 16th international conference on World Wide Web (WWW 2007)*. (Banff, Alberta, Canada). New York, NY, USA: ACM, 2007, pages 677–686. ISBN: 978-1-59593-654-7. (Cited on page 38).

[Kne08a] M. Knechtel. "Access restriction inside ontologies." In: *Proceedings of the 1st Internet of Services Doctoral Symposium 2008 at International Conference on Interoperability of Enterprise Systems and Applications (I-ESA 2008)*. Edited by R. Ruggaber. Volume 374. CEUR Workshop Proceedings. 2008.

[Kne08b] M. Knechtel. "Access rights and collaborative ontology integration for reuse across security domains." In: *Proceedings of the ESWC 2008 Ph.D. Symposium*. Edited by P. Cudré-Mauroux. Volume 358. CEUR Workshop Proceedings. *Best Poster Award.* 2008, pages 36–40.

[KP10a] M. Knechtel and R. Peñaloza. "A Generic Approach for Correcting Access Restrictions to a Consequence." In: *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*. Edited by L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache. Volume 6088. Lecture Notes in Computer Science. 2010, pages 167–182. (Cited on pages 61, 85).

[KP10b] M. Knechtel and R. Peñaloza. "Correcting Access Restrictions to a Consequence." In: *Proceedings of the 23rd International Workshop on Description Logics (DL 2010)*. Edited by V. Haarslev, D. Toman, and G. Weddell. Volume 573. CEUR Workshop Proceedings. 2010, pages 220–231. (Cited on pages 61, 85).

[KS08] M. Knechtel and D. Schuster. "Semantische Integration und Wiederverwendung von Produktontologien für offene Marktplätze im Web." In: *Tagungsband des Gemeinschaften in Neuen Medien Workshops (GeNeMe 2008)*. Edited by K. Meißner and M. Engelien. In German. 2008, pages 177–188. (Cited on page 89).

[KS10] M. Knechtel and H. Stuckenschmidt. "Query-Based Access Control for Ontologies." In: *Proceedings of the 4th International Conference on Web Reasoning and Rule Systems (RR 2010)*. Edited by P. Hitzler and T. Lukasiewicz. Volume 6333. Lecture Notes in Computer Science. 2010, pages 73–87. (Cited on page 89).

[Lam71] B. Lampson. "Protection." In: *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*. 1971, pages 437–443. (Cited on page 31).

[Les+08] M.-J. Lesot, O. Couchariere, B. Bouchon-Meunier, and J.-L. Rogier. "Inconsistency Degree Computation for Possibilistic Description Logic: An Extension of the Tableau Algorithm." In: *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS 2008)*. IEEE Computer Society Press, 2008, pages 1–6. (Cited on page 63).

[LS00] C. Lutz and U. Sattler. "Mary likes all Cats." In: *Proceedings of the 13rd International Workshop on Description Logics (DL 2000)*. Edited by F. Baader and U. Sattler. CEUR Workshop Proceedings 33. Aachen, Germany: RWTH Aachen, 2000, pages 213–226. (Cited on page 49).

[Mey+06] T. Meyer, K. Lee, R. Booth, and J. Z. Pan. "Finding Maximally Satisfiable Terminologies for the Description Logic $\mathcal{ALC}$." In: *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*. AAAI Press/The MIT Press, 2006. (Cited on pages 25, 63, 77).

[Min81] M. Minsky. "A Framework for Representing Knowledge." In: *Mind Design*. Edited by J. Haugeland. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [BL85]. The MIT Press, 1981. (Cited on page 18).

[MPSP09] B. Motik, P. F. Patel-Schneider, and B. Parsia. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-syntax/`. World Wide Web Consortium, Oct. 2009. (Cited on pages 2, 18, 19, 23, 24, 41).

[MSH07] B. Motik, R. Shearer, and I. Horrocks. "Optimized Reasoning in Description Logics Using Hypertableaux." In: *Proceedings of the 21st Conference on Automated Deduction (CADE 2007)*. Edited by F. Pfenning. Volume 4603. Lecture Notes in Computer Science. 2007, pages 67–83. (Cited on page 19).

[Obe+09] D. Oberle, N. Bhatti, S. Brockmans, M. Niemann, and C. Janiesch. "Countering Service Information Challenges in the Internet of Services." In: *Business & Information Systems Engineering* 1.5 (2009), pages 370–390. (Cited on page 4).

[OSM00] S. Osborn, R. Sandhu, and Q. Munawer. "Configuring role-based access control to enforce mandatory and discretionary access control policies." In: *ACM Transactions on Information and System Security* 3.2 (2000), pages 85–106. ISSN: 1094-9224. (Cited on page 34).

[QA03] L. Qin and V. Atluri. "Concept-level access control for the Semantic Web." In: *Proceedings of the 2003 ACM workshop on XML security (XMLSEC 2003)*. (Fairfax, Virginia). New York, NY, USA: ACM, 2003, pages 94–103. ISBN: 1-58113-777-X. (Cited on page 91).

[QP08] G. Qi and J. Z. Pan. "A Tableau Algorithm for Possibilistic Description Logic." In: *Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008)*. Edited by J. Domingue and C. Anutariya. Volume 5367. Lecture Notes in Computer Science. Springer-Verlag, 2008, pages 61–75. (Cited on page 63).

[QPJ07] G. Qi, J. Z. Pan, and Q. Ji. "Extending Description Logics with Uncertainty Reasoning in Possibilistic Logic." In: *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007)*. Edited by K. Mellouli. Volume 4724. Lecture Notes in Computer Science. Springer-Verlag, 2007, pages 828–839. (Cited on page 64).

[Qui67] M. Quillian. "Word concepts: A theory and simulation of some basic capabilities." In: *Behavioral Science* 12 (1967). Republished in [BL85], pages 410–430. (Cited on page 18).

[Rei87] R. Reiter. "A Theory of Diagnosis from First Principles." In: *Artificial Intelligence* 32.1 (1987), pages 57–95. (Cited on pages 25, 68, 96).

[RKH08] S. Rudolph, M. Krötzsch, and P. Hitzler. "All Elephants are Bigger than All Mice." In: *Proceedings of the 21st International Workshop on Description Logics (DL 2008)*. 2008. (Cited on pages 42, 48).

[RS07] A. A. Rudi Studer Stephan Grimm, editor. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007. ISBN: 978-3-540-70893-3. (Cited on page 21).

[San93] R. S. Sandhu. "Lattice-Based Access Control Models." In: *IEEE Computer* 26.11 (1993), pages 9–19. ISSN: 0018-9162. (Cited on pages 32, 33, 34).

[Sar08] S. Sarawagi. "Information Extraction." In: *Foundations and Trends in Databases* 1.3 (2008), pages 261–377. (Cited on page 20).

[SC03] S. Schlobach and R. Cornet. "Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies." In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2003)*. Edited by G. Gottlob and T. Walsh. Morgan Kaufmann, 2003, pages 355–362. (Cited on pages 25, 63, 68, 77).

[Sch08] S. Schenk. "On the Semantics of Trust and Caching in the Semantic Web." In: *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*. 2008, pages 533–549. (Cited on page 104).

[SFK00] R. Sandhu, D. Ferraiolo, and R. Kuhn. "The NIST model for role-based access control: towards a unified standard." In: *Proceedings of the 5th ACM workshop on Role-based access control (RBAC 2000)*. (Berlin, Germany). New York, NY, USA: ACM, 2000, pages 47–63. ISBN: 1-58113-259-X. (Cited on pages 2, 33, 34, 41).

[SGC79] L. K. Schubert, R. G. Goebel, and N. J. Cercone. "The Structure and Organization of a Semantic Net for Comprehension and Inference." In: *Associative Networks: Representation and Use of Knowledge by Computers*. Edited by N. V. Findler. Academic Press, 1979, pages 121–175. (Cited on page 18).

[SHV01] G. Saunders, M. Hitchens, and V. Varadharajan. "Role-based access control and the access control matrix." In: *ACM SIGOPS Operating Systems Review* 35.4 (2001), pages 6–20. ISSN: 0163-5980. (Cited on page 34).

[Sir+07] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. "Pellet: A practical OWL-DL reasoner." In: *Journal of Web Semantics* 5.2 (2007), pages 51–53. (Cited on pages 19, 25, 104).

[SP07] E. Sirin and B. Parsia. "SPARQL-DL: SPARQL Queries for OWL-DL." In: *Proceedings of the 3rd International Workshop on OWL: Experiences and Directions (OWLED 2007)*. 2007. (Cited on pages 92, 93).

[Stu96] G. Stumme. "Attribute exploration with background implications and exceptions." In: *Proceedings of the Annual Conference of the GfKl (GfKl 1995)*. Edited by H.-H. Bock and W.Polasek. Springer, 1996, pages 457–469. (Cited on pages 53, 56).

[Sun08] B. Suntisrivaraporn. "Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies." Available at `http://nbn-resolving.de/urn:nbn:de:bsz:14-ds-1233830966436-59282`. PhD thesis. Technische Universität Dresden, 2008. (Cited on pages 24, 25, 64, 68, 69, 72, 80, 85, 104, 109).

[Tan10] V. Tannen. "Provenance for database transformations." In: *Proceedings of the 13th International Conference on Extending Database Technology (EDBT 2010)*. 2010, page 1. (Cited on pages 63, 64).

[TH05] D. Tsarkov and I. Horrocks. "Ordering Heuristics for Description Logic Reasoning." In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*. (Edinburgh, Scotland). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, pages 609–614. (Cited on page 19).

[THE10] THESEUS Joint Research. *THESEUS Application Scenario PROCESSUS*. `http://www.theseus.joint-research.org/processus-en/`. 2010. (Cited on page 4).

[Zha+05] C. Zhao, N. Heilili, S. Liu, and Z. Lin. "Representation and Reasoning on RBAC: A Description Logic Approach." In: *Proceedings of the 2nd International Colloquium Theoretical Aspects of Computing (ICTAC 2005)*. 2005, pages 381–393. (Cited on page 38).