Formal Concept Analysis Methods for Description Logics

Dissertation

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

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
M.Sc. Barış Sertkaya
geboren am 3. Oktober 1978 in Olukbaşı, Türkei

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

Prof. Dr.rer.nat. Bernhard Ganter,
Technische Universität Dresden

Prof. Dr.rer.nat. habil. Ulrike Sattler,
University of Manchester

Tag der Verteidigung: 15. November 2007

Dresden, im Juli 2008

Acknowledgements

I am deeply indebted to my supervisors Franz Baader and Bernhard Ganter whose invaluable advices and ideas brought this work to life. Without their encouragement and friendly support this thesis would have never been finished. I would like to thank Ulrike Sattler for her valuable ideas and comments on the parts of this thesis, for accepting to review it, and also for her excellent proof-reading. I would also like to thank Miki Hermann for his comments, and the expertise he provided on the parts of this thesis.

Many thanks to the members of the Automata Theory group at TU Dresden for the friendly environment they provided, for the cake sessions, for the inspiring working atmosphere and for sharing the last four years. Among them my dear officemates Maja Miličić and Felix Distel deserve special thanks. Thanks to Maja for sharing a friendly office for four years, for patience, for picnics and the WG parties.

Thanks to Ozan Kahramanoğulları, Sebastian Rudolph, Maja and Felix for their careful proof-reading the parts of a preliminary version of this thesis. Also thanks to Ozan for the beers and Friday night meetings. Many thanks to the friends from my home country Turkey that stayed close despite the distance.

The first three years of this work has been financially supported by the German Research Foundation DFG under the grant GRK 334/3. Freunde und Förderer der Informatik an der TU Dresden and the EU Projects TONES and Semantic Mining have partially supported the last year. I would like to thank Franz Baader for finding me the financial support.

Special thanks to my parents and my sisters for the constant support and warmth they gave me during the preparation of this work. I am most grateful to my beloved Kathrin for her continuous support and precious love. She was always standing by me in every respect: "*Danke Kathrinim!*" Finally, thanks to Levin Can for the smiles, for making life enjoyable, and for keeping me awake: "*baba kalk, oyun oynayalım!*"...

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The notion of a concept as a collection of objects sharing certain properties, and the notion of a conceptual hierarchy are fundamental to both Formal Concept Analysis (FCA) and Description Logics (DLs). However, the ways concepts are described and obtained differ significantly between these two research areas. In DLs, the relevant concepts of the application domain are formalized by so-called concept descriptions, which are expressions built from unary predicates (that are called atomic concepts), and binary predicates (that are called atomic roles) with the help of the concept constructors provided by the DL language. Then in a second step, these concept descriptions are used to describe properties of individuals occurring in the domain, and the roles are used to describe relations between these individuals. On the other hand, in FCA, one starts with a so-called formal context, which in its simplest form is a way of specifying which attributes are satisfied by which objects. A formal concept of such a context is a pair consisting of a set of objects called extent, and a set of attributes called intent such that the intent consists of exactly those attributes that the objects in the extent have in common, and the extent consists of exactly those objects that share all attributes in the intent.

There are several differences between these approaches. First, in FCA one starts with a purely extensional description of the application domain, and then derives the formal concepts of this specific domain, which provide a useful structuring. In a way, in FCA the intensional knowledge is obtained from the extensional part of the knowledge. On the other hand, in DLs the intensional definition of a concept is given independently of a specific domain (interpretation), and the description of the individuals is only partial. Second, in FCA the properties are atomic, and the intensional description of a formal concept (by its intent) is just a conjunction of such properties. DLs usually provide a richer language for the intensional definition of concepts, which can be seen as an expressive, yet decidable sublanguage of first-order predicate logic.

Despite these differences, there have been several attempts to bridge the gap between these two formalisms, and attempts to apply methods from one field to the other. For example, there have been efforts to enrich FCA with more complex properties similar to concept constructors in DLs [Zic91, PS99, Pre00, FRS05, RHNV07]. On the other hand, DL research has benefited from FCA methods to solve some problems encountered in knowledge representation using DLs [Baa95, Stu96b, BM00, BS04, Rud04, BST04a, Rud06, Ser06, BST07, BGSS07b]. In the present work, we are interested in the latter approach, namely applications of FCA methods in DLs.

1.1 Description Logics

Description Logics are a class of knowledge representation formalisms that are used to represent the terminological knowledge of an application domain in a structured way. They originated as a result of the efforts to give a logic-based semantics to earlier knowledge representation formalisms called *semantic networks* and *frames*. Semantic networks [Qui67] were developed with the aim to represent knowledge and do reasoning by means of network-shaped cognitive structures. Frame systems [Min81], on the other hand, relied upon so-called "frames" for the same purposes. Although there are significant differences between semantic networks and frames, they can both be seen as network structures where the sets of individuals and relations among these individuals are represented in the structure of the network. From a practical point of view, these formalisms were easy to understand and use. Unfortunately, on the theoretical side they suffered from their lack of a precise semantics. As a result, different systems based on these formalisms sometimes behaved differently upon the same input. Efforts to overcome this problem led to the recognition that first-order logic could be used to provide semantics to frame systems [Hay79]. Later, in [BL85], it was realized that frames and semantic networks actually did not need the full power of first-order logic, but they could be expressed in fragments of it where reasoning is decidable. This realization initiated the research on so-called *KL-ONE-like representation languages*, which were fragments of first-order logic, and on developing specialized algorithms for reasoning with these languages. Over time, the name KL-ONE-like languages first changed to *concept languages*, then to *terminological logics*, and finally to *Description Logics (DLs)*. Since their introduction, DLs have been used in various application domains such as medical informatics, software engineering, configuration of technical systems, natural language processing, databases and web-based information systems. But their most notable success so far is the adoption of the DL-based language OWL[1] [HPSvH03] as the standard

[1] Web Ontology Language. See `http://www.w3.org/TR/owl-features`

ontology language for the semantic web [BLHL01].

Syntax

In DLs, one formalizes the relevant notions of an application domain by *concept descriptions*. A concept description is an expression built from atomic concepts, which are unary predicates, and atomic roles, which are binary predicates, by using the concept constructors provided by the particular DL language in use. The set of atomic concepts is usually represented with N_C, and the set of atomic roles is usually represented with N_R. DL languages are identified with the concept constructors they allow. The language allowing for the constructors $\sqcap$ (conjunction), $\sqcup$ (disjunction), $\neg$ (negation), $\forall$ (value restriction) and $\exists$ (existential restriction) is called $\mathcal{ALC}$ [SSS91].

Typically, a *DL knowledge base* consists of a *terminological box* (*TBox*), which defines the terminology of an application domain, and an *assertional box* (*ABox*), which contains facts about a specific world. In its simplest form, a TBox is a set of *concept definitions* of the form $A \equiv C$ that assigns the concept name A to the concept description C. The concept names occurring on the left-hand side of a concept definition are called *defined concepts*, and the others are called *primitive concepts*. We say that a TBox is *acyclic* if no concept definition directly or indirectly refers to the name it defines. We call a finite set of *general concept inclusion (GCI)* axioms a *general TBox*. A GCI is an expression of the form $C \sqsubseteq D$, where C and D are two possibly complex concept descriptions. It states a subconcept/superconcept relationship between the two concept descriptions. An ABox is a set of *concept assertions* of the form $C(a)$, which means that the individual a is an instance of the concept C, and *role assertions* of the form $R(a, b)$, which means that the individual a is in R-relation with individual b. Let us demonstrate these notions on a toy knowledge base:

Example 1.1.1. Assume our set of atomic concepts N_C contains the concepts Country, Land and Ocean, and the set of atomic roles N_R contains the role hasBorderTo, and we are using a DL that provides $\sqcap$, $\forall$ and $\exists$. The following TBox contains the definition of a landlocked country, which is a country that only has borders on land, and the definition of an ocean country that has a border to an ocean.

$$\begin{aligned}\mathcal{T} := \{\mathsf{LandlockedCountry} &\equiv \mathsf{Country} \sqcap \forall \mathsf{hasBorderTo.Land} \\ \mathsf{OceanCountry} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasBorderTo.Ocean}\}\end{aligned}$$

The following ABox states the facts about the individuals *Portugal*, *Austria*, and *Atlantic Ocean*.

$$\begin{aligned}\mathcal{A} := \{&\mathsf{LandlockedCountry}(\mathit{Austria}), \\ &\mathsf{Country}(\mathit{Portugal}), \\ &\mathsf{Ocean}(\mathit{Atlantic\ Ocean}), \\ &\mathsf{hasBorderTo}(\mathit{Portugal}, \mathit{Atlantic\ Ocean})\}\end{aligned}$$

Semantics

The meaning of DL concepts is given by means of an *interpretation* $\mathcal{I}$, which is a tuple consisting of a *domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$. The interpretation function maps every concept occurring in the TBox to a subset of the domain, every role to a binary relation on the domain, and every individual name occurring in the ABox to an element of the domain. The meaning of complex concept descriptions is given inductively based on the constructors used in the concept description.

For instance, the concept description $\mathsf{Country} \sqcap \exists \mathsf{hasBorderTo.Ocean}$ is interpreted as the intersection of the set of countries and the set of elements of the domain that have a border to an ocean. We say that an interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ if it satisfies all concept definitions in $\mathcal{T}$, i.e., for every concept definition $A \equiv C$ in $\mathcal{T}$, it maps A and C to the same subset of the domain. Similarly, we say that $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$, if it satisfies all concept and role assertions in $\mathcal{A}$, i.e., for every concept assertion $A(a)$ in $\mathcal{A}$, the interpretation of a is an element of the interpretation of A, and for every role assertion $r(a, b)$ the interpretation of r contains the pair consisting of the interpretations of a and b. The semantics of ABoxes is the *open-world semantics*, i.e., absence of information about an individual is not interpreted as negative information, but it only indicates lack of knowledge.

Inferences

In an application, once we get a description of the application domain using DLs as described above, we can make inferences, i.e., deduce implicit consequences from the explicitly represented knowledge. The basic inference on concept descriptions is *subsumption*. Given two concept descriptions C and D, the subsumption problem $C \sqsubseteq D$ is the problem of checking whether the concept description D is more general than the concept description C. In other words, it is the problem of determining whether the first concept always, i.e., in every interpretation denotes a subset of the set denoted by the second one. We say that C is subsumed by D w.r.t. a TBox $\mathcal{T}$, if in every model of $\mathcal{T}$, D is more general than C, i.e., the interpretation of C is a subset of the interpretation of D. We denote this as $C \sqsubseteq_{\mathcal{T}} D$. For instance, in Example 1.1.1 above, the concepts $\mathsf{LandlockedCountry}$ and $\mathsf{OceanCountry}$ are both trivially subsumed by the concept $\mathsf{Country}$.

Another typical inference on concept descriptions is *satisfiability*, which is the problem of checking whether there is an interpretation that interprets a given concept description as a nonempty set. We say that a concept is satisfiable w.r.t. a TBox if the TBox has a model in which the interpretation of the concept is not empty. In fact, the satisfiability problem can be reduced to the subsumption problem. A concept is unsatisfiable if and only if it is subsumed by bottom, which is the empty concept.

The typical inference problems for ABoxes are instance checking and consistency. *Consistency* of an ABox w.r.t. a TBox is the problem of checking whether the ABox and the TBox have a common model. *Instance checking* w.r.t. a TBox and an ABox is the problem of deciding whether the interpretation of a given individual is an element of the interpretation of a given concept in every common model of the TBox and the ABox. For instance, in Example 1.1.1 above, it follows from $\mathcal{T}$ and $\mathcal{A}$ that *Portugal* is an ocean country, although $\mathcal{A}$ does not contain the assertion $\mathsf{OceanCountry}(Portugal)$. Modern *DL systems*, i.e., knowledge representation systems based on DLs, like FaCT [Hor98] (its successor FaCT++ [TH06]), RACER [HM01b] (its successor RACERPRO), Pellet [SP04] and KAON2 [Mot06] provide their users with inference services that solve these inference problems, which are also known as *standard inferences*.

In Chapter 2 we are going to give a more formal introduction to DLs. A broader introduction can be found in [NB03, BN03].

1.2 Formal Concept Analysis

Formal Concept Analysis (FCA) is a field of applied mathematics that aims to formalize the notions of a concept and a conceptual hierarchy by means of mathematical tools. It facilitates the use of mathematical reasoning for conceptual data analysis and knowledge processing. FCA emerged around 1980 as a result of the attempts to restructure mathematical order and lattice theory [Wil82]. Later on it has been developed as a subfield of applied mathematics based on the mathematization of concepts and conceptual hierarchies.

Formal context and formal concept

In FCA, one represents data in the form of a *formal context*, which in its simplest form is a way of specifying which attributes are satisfied by which objects. A formal context is usually visualized as a cross table, where the rows represent the objects, and the columns represent the attributes of the context. A cross in column m of row g means that the object g has the attribute m, and the absence of a cross means that g does not have attribute m. A formal context is usually denoted by $\mathbb{K} = (G, M, I)$ where G is the set of objects, M is the set of attributes, and I is the relation between the objects and the attributes. Let us give an example of a formal context.

Example 1.2.1. The formal context $\mathbb{K}_{countries}$ in Table 1.1 below shows the membership status of some countries to some international organisations, and whether they have nuclear weapons or not. The attributes of the context

are, *has nuclear weapons*,[2] *UN Security Council permanent member*, *UN Security Council temporary member*,[3] *G8 member*, *EU member*,[3] and *UN member*. ◇

$\mathbb{K}_{countries}$	has nuclear weapons	UN Security Council permanent	UN Security Council temporary	G8 mem.	EU mem.	UN mem.
USA	X	X		X		X
Germany				X	X	X
France	X	X		X	X	X
UK	X	X		X	X	X
Turkey						X
Qatar			X			X
Italy			X	X	X	X

Table 1.1: Formal context $\mathbb{K}_{countries}$.

Given such a formal context, the first step for analyzing this context is usually computing the *formal concepts* of this context, which are "natural clusterings" of the data in the context. A formal concept is a pair consisting of an object set A and an attribute set B such that the objects in A share the attributes in B, and B consists of exactly those attributes that the objects in A have in common. The object set A is called the *extent*, and the attribute set B is called the *intent* of the formal concept (A, B). Formal concepts of the context given in Example 1.2.1 are listed below.

Example 1.2.2. The formal context $\mathbb{K}_{countries}$ in Example 1.2.1 has 8 formal concepts. In order to avoid redundant text while listing them, here we are going to use the following abbreviations for the attribute names given in the order in Example 1.2.1 above: *nw*, *SCp*, *SCt*, *G8*, *EU*, *UN*. The formal concepts of $\mathbb{K}_{countries}$ are:

- ({}, {nw, SCp, SCt, G8, EU , UN })
- ({Italy}, {SCt, G8, EU, UN})
- ({UK, France}, {nw, SCp, G8, EU, UN})
- ({Qatar, Italy}, {SCt, UN})
- ({Germany, UK, France, Italy}, {EU, G8, UN})
- ({USA, France, UK}, {nw, SCp, G8, UN})
- ({USA, Germany, France, UK, Italy}, {G8, UN})

[2] countries that have the internationally recognised status conferred by the NPT. Source: `http://en.wikipedia.org/wiki/List_of_countries_with_nuclear_weapons`

[3] as of June 2007

- ({Turkey, USA, Germany, Qatar, France, UK, Italy}, {UN}).

◇

Concept lattice

Once all formal concepts of a context are obtained, one orders them w.r.t. the inclusion of their extents (equivalently, inverse inclusion of their intents). This ordering gives a complete lattice, called the *concept lattice* of the context. A concept lattice contains all information represented in a formal context, i.e., we can easily read off the attributes, objects and the incidence relation of the underlying context. Moreover, visualization of a concept lattice makes it easier to see formal concepts of a context and interrelations among them. Thus it helps to understand the structure of the data in the formal context, and to query the knowledge represented in the formal context.

The nodes of a concept lattice represent the formal concepts of the underlying context. In order to improve readability of the lattice, we avoid writing down the extent and intent of every single node. Instead, we label the nodes with attribute and object names in such a way that every name appears only once in the lattice. In this labelling, the intent of the formal concept corresponding to a node can be determined by the attribute names that can be reached by the ascending lines, and its extent can be determined by the object names that can be reached by the descending lines. For instance consider the concept lattice in Figure 1.1 that results from the formal context $\mathbb{K}_{countries}$ in Example 1.2.1: the intent of the formal concept marked with the attribute name *EU member* and with the object name *Germany* is {*EU member, G8 member, UN member*}, and its extent is {*Germany, France, UK, Italy*}.

Given a formal context, one other common method to analyze it is to find (a base) of the implications between attributes of this context. Implications between attributes are constraints between attributes that hold in the given context. They are statements of the form "*every object that satisfies the attributes* $m_{i1}, \ldots, m_{ik}$ *also satisfies the attributes* $m_{j1}, \ldots, m_{jl}$". The concept lattice of a context can be reconstructed from its implications. Conversely, the implications between the attributes of a formal context can be read off from the concept lattice.

Attribute exploration

Attribute exploration [Gan84, GW99] is a knowledge acquisition method of FCA that is used to acquire knowledge from a domain expert by asking successive questions. In many applications where the formal context is not explicitly given, but it is only "known" to a domain expert, it has proved to

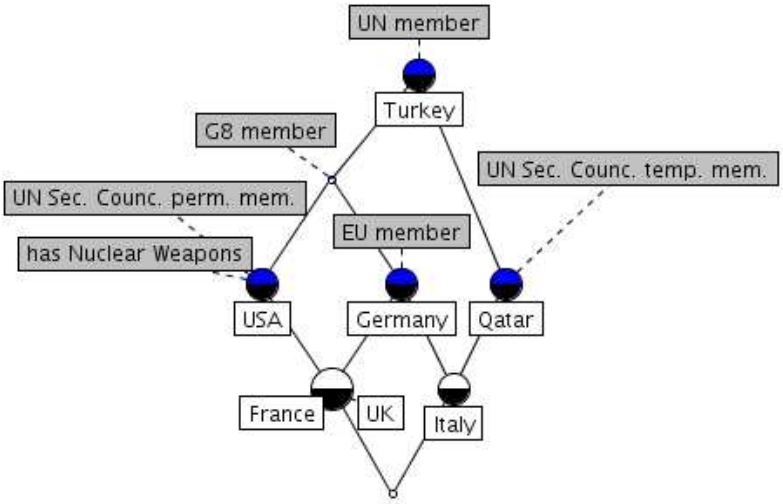

Figure 1.1: Concept lattice of $\mathbb{K}_{countries}$

be a successful method for efficiently capturing the expert's knowledge. Attribute exploration asks the expert questions of the form "*is it true that objects having attributes* $m_{i1}, \ldots, m_{ik}$ *also have the attributes* $m_{j1}, \ldots, m_{jl}$*?*". When such a question is asked, the expert is expected to either confirm the question, in which case a new implication of the application domain is found, or reject it. If the expert rejects such a question, she is expected to give a counterexample, i.e., an object that has all the attributes $m_{i1}, \ldots, m_{ik}$ but lacks at least one of $m_{j1}, \ldots, m_{jl}$. This counterexample is then added to the application domain as a new object, and the next question is asked. What makes attribute exploration an attractive method for capturing expert knowledge is that it guarantees to make best use of the expert's answers, and to ask the minimum possible number of questions that suffices to acquire complete knowledge about the application domain.

1.3 Existing work on DLs and FCA

DLs and FCA follow significantly different methodologies for obtaining and defining the relevant concepts of a domain. In DLs, the (intensional) definition of a concept is given independently of a specific interpretation, and the description of the objects is only partial. In FCA, one starts with a purely extensional description of the domain, and then derives the formal concepts of this specific domain. In a way, in FCA the intensional knowledge is obtained from the extensional part of the knowledge. Despite these differences, there have been several attempts towards bridging the gap between these two formalisms, and attempts to apply methods from one field to the other.

It is possible to collect these attempts roughly under two categories:

- efforts to enrich the language of FCA by borrowing constructors from DL languages [Zic91, PS99, Pre00, FRS05, RHNV07]
- efforts to employ FCA methods in the solution of problems encountered in knowledge representation with DLs [Baa95, Stu96b, BM00, Rud04, Rud06]

Below we are going to discuss some of these efforts briefly.

1.3.1 Enriching FCA with DL constructors

Theory-driven logical scaling

In [PS99], Prediger and Stumme have used DLs in *Conceptual Information Systems*, which are data analysis tools based on FCA. They can be used to extract data from a relational database and to store it in a formal context by using so-called *conceptual scales*. Prediger and Stumme have combined DLs with attribute exploration in order to define a new kind of conceptual scale. In this approach, DLs provide a rich language to specify which FCA attributes cannot occur together, and a DL reasoner is used during the attribute exploration process as an expert to answer the implication questions, and to provide a counterexample whenever the implication does not hold.

Terminological attribute logic

In [Pre00], Prediger has worked on introducing logical constructors into FCA. She has enriched FCA with relations, existential and universal quantifiers, and negation, obtaining a language like the DL $\mathcal{ALC}$, which she has called *terminologische Merkmalslogik* (*terminological attribute logic*[4]). In the same work she has also presented applications of her approach in enriching formal contexts with new knowledge, applications in many valued formal contexts, and applications for so-called *scales*, which are formal contexts that are used to obtain a standard formal context from a many valued formal context.

Relational concept analysis

In [RHNV07], Rouane et al. have presented a combination of FCA and DLs that is called *relational concept analysis*. It is an adaptation of FCA that is intended for analyzing objects described by relational attributes in data mining. The approach is based on a collection of formal contexts called *relational context family* and relations between these contexts. The relations

[4]This translation is ours.

between the contexts are binary relations between pairs of object sets that belong to two different contexts. Processing these contexts and relations with relational concept analysis methods yields a set of concept lattices (one for each input context) such that the formal concepts in different lattices are linked by relational attributes, which are similar to roles in DLs, or associations in UML. One distinguishing feature of this approach from the other efforts that introduce relations into FCA is that the formal concepts and relations between formal concepts of different contexts can be mapped into concept descriptions in a sublanguage of $\mathcal{ALE}$, which is called $\mathcal{FL}^-\mathcal{E}$ in [RHNV07]. $\mathcal{FL}^-\mathcal{E}$ allows for conjunction, value restriction, existential restriction, and top and bottom concepts. In this approach, after the formal concepts and relations have been obtained and mapped into $\mathcal{FL}^-\mathcal{E}$ concept descriptions, DL reasoning is used to classify and check the consistency of these descriptions.

1.3.2 Applying FCA methods in DLs

Subsumption hierarchy of conjunctions of DL concepts

In [Baa95], Baader has used FCA for an efficient computation of an extended subsumption hierarchy of a set of DL concepts. More precisely, he used attribute exploration for computing the subsumption hierarchy of all conjunctions of a set of DL concepts. The main motivation for this work was to determine the interaction between defined concepts, which might not easily be seen by just looking at the subsumption hierarchy of defined concepts. In order to explain this, the following example has been given: assume that the defined concept NoDaughter stands for those people who have no daughters, NoSon stands for those people who have no sons, and NoSmallChild stands for those people who have no small children. Obviously, there is no subsumption relationship between these three concepts. On the other hand, the conjunction NoDaughter $\sqcap$ NoSon is subsumed by NoSmallChild, i.e., if an individual a belongs to NoSon and NoDaughter, it also belongs to NoSmallChild. However, this cannot be derived from the information that a belongs to NoSon and NoDaughter by just looking at the subsumption hierarchy. This small example demonstrates that runtime inferences concerning individuals can be made faster by precomputing the subsumption hierarchy not only for defined concepts, but also for all conjunctions of defined concepts.

To this purpose, Baader defined a formal context whose attributes were the defined DL concepts, and whose objects were all possible counterexamples to subsumption relationships, i.e., interpretations together with an element of the interpretation domain. This formal context has the property that its concept lattice is isomorphic to the required subsumption hierarchy, namely the subsumption hierarchy of conjunctions of the defined DL con-

cepts. However, this formal context has the disadvantage that a standard subsumption algorithm can not be used as expert for this context within attribute exploration. In order to overcome this problem, the approach was reconsidered in [BS04] and a new formal context that has the same properties but for which a usual subsumption algorithm could be used as expert was introduced. We are going to talk about this approach in more detail in Section 4.6.

Subsumption hierarchy of conjunctions and disjunctions of DL concepts

In [Stu96b], Stumme has extended the abovementioned subsumption hierarchy further with disjunctions of DL concepts. More precisely, he presented how the complete lattice of all possible combinations of conjunctions and disjunctions of the concepts in a DL TBox can be computed by using FCA. To this aim, he used another knowledge acquisition tool of FCA instead of attribute exploration, namely *distributive concept exploration* [Stu98]. In the lattice computed by this method, the supremum of two DL concepts in the lattice corresponds to the disjunction of these concepts.

Subsumption hierarchy of least common subsumers

In [BM00] Baader and Molitor have used FCA for supporting bottom-up construction of DL knowledge bases. In the bottom-up approach, the knowledge engineer does not directly define the concepts of her application domain, but she gives typical examples of a concept, and the system comes up with a concept description for these examples. The process of computing such a concept description consists of first computing the *most specific concepts* that the given examples belong to, and then computing the *least common subsumer* of these concepts. Here the choice of examples is crucial for the quality of the resulting concept description. If the examples are too similar, the resulting concept description will be too specific; conversely, if they are too distinct, the resulting concept description will be too general. In order to overcome this, Baader and Molitor have used attribute exploration for computing the subsumption hierarchy of all least common subsumers of a given set of concepts. In this hierarchy one can easily see the position of the least concept description that the chosen examples belong to, and decide whether these examples are appropriate for obtaining the intended concept description. However, there may be exponentially many least common subsumers, and depending on the DL in use, both the least common subsumer computation and subsumption test can be expensive operations. The use of attribute exploration provides us with complete information on how this hierarchy looks like without explicitly computing all least common subsumers and classifying them. We are going to talk about this approach

in more detail in Section 4.7.

Relational exploration

In his Ph.D thesis [Rud06], Rudolph has combined DLs and FCA for acquiring complete relational knowledge about an application domain. In his approach, which he calls *relational exploration*, he uses DLs for defining FCA attributes, and FCA for refining DL knowledge bases. More precisely, DLs makes use of the interactive knowledge acquisition method of FCA, and FCA benefits from DLs in terms of expressing relational knowledge.

In [Rud04, Rud06], Rudolph uses the DL $\mathcal{FLE}$ for this purpose, which is the DL that allows for the constructors conjunction, existential restriction, and value restriction. In his previous work [Rud03], he uses the DL $\mathcal{EL}$, which allows for the constructors conjunction and existential restriction. In both cases, he defines the semantics by means of a special pair of formal contexts called *binary power context family*, which are used for expressing relations in FCA. Binary power context families have also been used for giving semantics to *conceptual graphs*. In order to collect information about the formulae expressible in $\mathcal{FLE}$, in [Rud04, Rud06] he defines a formal context called *$\mathcal{FLE}$-context*. The attributes of this formal context are $\mathcal{FLE}$-concept descriptions, and the objects are the elements of the domain over which these concept descriptions are interpreted. In this context, an object g is in relation with an attribute m if and only if g is in the interpretation of m. Thus, an implication holds in this formal context if and only if in the given model the concept description resulting from the conjunction of the attributes in the premise of the implication is subsumed by the concept description formed from the conclusion. This is how implications in $\mathcal{FLE}$-contexts give rise to subsumption relationships between $\mathcal{FLE}$ concept descriptions.

In order to obtain *complete* knowledge about the subsumption relationships in the given model between arbitrary $\mathcal{FLE}$ concepts, Rudolph gives a multi-step exploration algorithm. In the first step of the algorithm, he starts with an $\mathcal{FLE}$-context whose attributes are the atomic concepts occurring in a knowledge base. In exploration step $i+1$, he defines the set of attributes as the union of the set of attributes from the first step and the set of concept descriptions formed by universally quantifying all attributes of the context at step i w.r.t. all atomic roles, and the set of concept descriptions formed by existentially quantifying all concept intents of the context at step i w.r.t all atomic roles. Rudolph points out that, at an exploration step, there can be some concept descriptions in the attribute set that are equivalent, i.e., attributes that can be reduced. To this aim, he introduces a method that he calls *empiric attribute reduction*. In principle, it is possible to carry out infinitely many exploration steps, which means that the algorithm will not terminate. In order to guarantee termination, Rudolph restricts the number

of exploration steps. After carrying out i steps of exploration, it is then possible to decide subsumption (w.r.t. the given model) between any $\mathcal{FLE}$ concept descriptions up to role depth i just by using the implication bases obtained as a result of the exploration steps. In addition, he also characterizes the cases where finitely many steps are sufficient to acquire complete information for deciding subsumption between $\mathcal{FLE}$ concept descriptions with arbitrary role depth. Rudolph argues that his method can be used to support the knowledge engineers in designing, building and refining DL ontologies.

1.4 Contributions

Our contribution to the DL research by means of FCA methods falls mainly under two topics: 1) supporting bottom-up construction of DL knowledge bases, 2) completing DL knowledge bases. In Section 1.4.1 we briefly describe the use of FCA in the former, and in Section 1.4.2 we briefly describe the use of FCA in the latter contribution. One other contribution of our work, which is not to the field of applying FCA methods to DLs, but to the FCA theory itself, is investigating the complexity of some decision and counting problems related to minimal generators of closed sets. Our contribution on this topic is briefly discussed in Section 1.4.3.

1.4.1 Supporting bottom-up construction

Traditionally, DL knowledge bases are built in a top-down manner, in the sense that first the relevant notions of the domain are formalized by concept descriptions, and then these concept descriptions are used to specify properties of the individuals occurring in the domain. However, this top-down approach is not always adequate. On the one hand, it might not always be intuitive which notions of the domain are the relevant ones for a particular application. On the other hand, even if this is intuitive, it might not always be easy to come up with a clear formal description of these notions, especially for a domain expert who is not an expert in knowledge engineering. In order to overcome this, in [BK98, BKM99] a new approach, called "bottom-up approach", was introduced for constructing DL knowledge bases. In this approach, instead of directly defining a new concept, the domain expert introduces several typical examples as objects, which are then automatically generalized into a concept description by the system. This description is then offered to the domain expert as a possible candidate for a definition of the concept. The task of computing such a concept description can be split into two subtasks:

- computing the most specific concepts of the given objects,
- and then computing the least common subsumer of these concepts.

The *most specific concept* (msc) of an object o is the most specific concept description C expressible in the given DL language that has o as an instance. The *least common subsumer (lcs)* of concept descriptions $C_1, \ldots, C_n$ is the most specific concept description C expressible in the given DL language that subsumes $C_1, \ldots, C_n$. The problem of computing the lcs and (to a more limited extent) the msc has already been investigated in the literature [CH94, FP96, BK98, BKM99, KM01a, KM01b, KB01, Baa03a, Baa03b, Baa03c].

The methods for computing the least common subsumer are restricted to rather inexpressive descriptions logics not allowing for disjunction (and thus not allowing for full negation). In fact, for languages with disjunction, the lcs of a collection of concepts is just their disjunction, and nothing new can be learned from building it. In contrast, for languages without disjunction, the lcs extracts the "commonalities" of the given collection of concepts. Modern DL systems like FaCT [Hor98] (its successor FaCT++ [TH06]), RACER [HM01b] (its successor RACERPRO), and Pellet [SP04] are based on very expressive DLs, and there exist large knowledge bases that use this expressive power and can be processed by these systems [RH97, SH00, HM01a]. In order to allow the user to re-use concepts defined in such existing knowledge bases and still support the user in defining new concepts with the bottom-up approach sketched above, we propose the following *extended bottom-up approach*: assume that there is a fixed *background terminology* defined in an expressive DL; e.g., a large ontology written by experts, which the user has bought from some ontology provider. The user then wants to extend this terminology in order to adapt it to the needs of a particular application domain. However, since the user is not a DL expert, he employs a less expressive DL and needs support through the bottom-up approach when building this user-specific extension of the background terminology. There are several reasons for the user to employ a restricted DL in this setting: first, such a restricted DL may be easier to comprehend and use for a non-expert; second, it may allow for a more intuitive graphical or frame-like user interface; third, to use the bottom-up approach, the lcs must exist and make sense, and it must be possible to compute it with reasonable effort.

To make this more precise, consider a background terminology (TBox) $\mathcal{T}$ defined in an expressive DL $\mathcal{L}_2$. When defining new concepts, the user employs only a sublanguage $\mathcal{L}_1$ of $\mathcal{L}_2$, for which computing the lcs makes sense. However, in addition to primitive concepts and roles, the concept descriptions written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$. Let us call such concept descriptions $\mathcal{L}_1(\mathcal{T})$-concept descriptions. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, we want to compute their lcs in $\mathcal{L}_1(\mathcal{T})$, i.e., the least $\mathcal{L}_1(\mathcal{T})$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. In [BST04a, BST07] we have considered the case where $\mathcal{L}_1$ is the DL $\mathcal{ALE}$ and $\mathcal{L}_2$ is the DL $\mathcal{ALC}$, and shown the following result:

- If $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox, then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept

descriptions always exists.

Unfortunately, the proof of this result does not yield a practical algorithm. Due to this, in [BST04a, BST07] we have developed a more practical approach. Assume that $\mathcal{L}_1$ is a DL for which least common subsumers (without background TBox) always exist. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can compute a common subsumer w.r.t. $\mathcal{T}$ by just ignoring $\mathcal{T}$, i.e., by treating the defined names in $C_1, \ldots, C_n$ as primitive and computing the lcs of $C_1, \ldots, C_n$ in $\mathcal{L}_1$. However, the common subsumer obtained this way will usually be too general. In the present work we present a practical method for computing "good" common subsumers w.r.t. background TBoxes, which may not be the *least* common subsumers, but which are better than the common subsumers computed by ignoring the TBox. As a tool, this method uses attribute exploration [Gan84] (possibly with background knowledge [Gan99, GK99, GK05]). The application of attribute exploration for this purpose is the main topic of Chapter 4.

1.4.2 Completing DL knowledge bases

The most notable success of DLs so far is due to the fact that they provide the logical underpinning of OWL [HPSvH03], the standard ontology language for the semantic web [BLHL01]. As a consequence of this standardization, several ontology editors like Protégé [KFNM04], and Swoop [KPS$^+$06] now support OWL, and ontologies written in OWL are employed in more and more applications. As the size of these ontologies grows, tools that support improving their quality become more important. The tools available until now use DL reasoning to detect inconsistencies and to infer consequences, i.e., implicit knowledge that can be deduced from the explicitly represented knowledge. There are also promising approaches that allow to pinpoint the reasons for inconsistencies and for certain consequences, and that help the ontology engineer to resolve inconsistencies and to remove unwanted consequences [SC03, KPSG06]. These approaches address the quality dimension of *soundness* of an ontology, both within itself (consistency) and w.r.t. the intended application domain (no unwanted consequences). In the present work, we consider a different quality dimension: *completeness*. We provide a basis for formally well-founded techniques and tools that support the ontology engineer in checking whether an ontology contains all the relevant information about the application domain, and to extend the ontology appropriately if this is not the case.

As already mentioned, a DL knowledge base (nowadays often called ontology) usually consists of two parts, the terminological part (TBox), which defines concepts and also states additional constraints (GCIs) on the interpretation of these concepts, and the assertional part (ABox), which describes individuals and their relationship to each other and to concepts. Given an

application domain and a DL knowledge base describing it, we can ask whether the knowledge base contains all the relevant information about the domain:

- Are all the relevant constraints that hold between concepts in the domain captured by the TBox?
- Are all the relevant individuals existing in the domain represented in the ABox?

As an example, consider the OWL ontology for human protein phosphatases that has been described and used in [WBH+05]. This ontology was developed based on information from peer-reviewed publications. The human protein phosphatase family has been well characterised experimentally, and detailed knowledge about different classes of such proteins is available. This knowledge is represented in the terminological part of the ontology. Moreover, a large set of human phosphatases has been identified and documented by expert biologists. These are described as individuals in the assertional part of the ontology. One can now ask whether the information about protein phosphatases contained in this ontology is complete: are all the relationships that hold among the introduced classes of phosphatases captured by the constraints in the TBox, or are there relationships that hold in the domain, but do not follow from the TBox? Are all possible kinds of human protein phosphatases represented by individuals in the ABox, or are there phosphatases that have not yet been included in the ontology or even not yet have been identified?

Such questions cannot be answered by an automated tool alone. Clearly, to check whether a given relationship between concepts—which does not follow from the TBox—holds in the domain, one needs to ask a domain expert, and the same is true for questions regarding the existence of individuals not described in the ABox. The rôle of the automated tool is to ensure that the expert is asked as few questions as possible; in particular, she should not be asked trivial questions, i.e., questions that could actually be answered based on the represented knowledge. In the above example, answering a non-trivial question regarding human protein phosphatases may require the biologist to study the relevant literature, query existing protein databases, or even to carry out new experiments. Thus, the expert may be prompted to acquire new biological knowledge.

The attribute exploration method of FCA has proved to be a successful knowledge acquisition method in various application domains. One of the earliest applications of this approach is described in [Wil82], where the domain is lattice theory, and the goal of the exploration process is to find, on the one hand, all valid relationships between properties of lattices (like being distributive), and, on the other hand, to find counterexamples to all the relationships that do not hold. To answer a query whether a certain

relationship holds, the lattice theory expert must either confirm the relationship (by using results from the literature or by carrying out a new proof for this fact), or give a counterexample (again, by either finding one in the literature or constructing a new one).

Although this sounds very similar to what is needed in our case, we cannot directly use this approach. The main reason is the open-world semantics of description logic knowledge bases. Consider an individual i from an ABox $\mathcal{A}$ and a concept C occurring in a TBox $\mathcal{T}$. If we cannot deduce from the TBox $\mathcal{T}$ and $\mathcal{A}$ that i is an instance of C, then we do not assume that i does not belong to C. Instead, we only accept this as a consequence if $\mathcal{T}$ and $\mathcal{A}$ imply that i is an instance of $\neg C$. Thus, our knowledge about the relationships between individuals and concepts is incomplete: if $\mathcal{T}$ and $\mathcal{A}$ imply neither $C(i)$ nor $\neg C(i)$, then we do not know the relationship between i and C. In contrast, classical FCA and attribute exploration assume that the knowledge about objects is complete: a cross in row g and column m of a formal context says that object g has attribute m, and the absence of a cross is interpreted as saying that g does not have m.

There has been some work on how to extend FCA and attribute exploration from complete knowledge to the case of partial knowledge [Gan99, BH00, Hol04a, Hol04b, BH05, Rud06], and how to evaluate formulas in formal contexts that do not contain complete information [Obi02]. However, this work is based on assumptions that are different from ours. In particular, they assume that the expert cannot answer all queries and, as a consequence, the knowledge obtained after the exploration process may still be incomplete and the relationships between concepts that are produced in the end fall into two categories: relationships that are valid no matter how the incomplete part of the knowledge is completed, and relationships that are valid only in some completions of the incomplete part of the knowledge. In contrast, our intention is to complete the knowledge base, i.e., in the end we want to have complete knowledge about these relationships. What may be incomplete is the description of individuals used during the exploration process. In Chapter 5 we introduce our variant of FCA that can deal with partial knowledge, describe an attribute exploration procedure that works in this setting, and present in detail how it can be used to complete DL knowledge bases.

1.4.3 On the generators of closed sets

In FCA, so-called *closed sets* play an important rôle. For example the concept intents of a formal context are the sets closed under the *closure operator* induced by this formal context. In addition, the sets closed under a set of implications are fundamental to the attribute exploration algorithm. Here we investigate a natural problem for closed sets, namely the problem of finding the generators of a closed set. More precisely, we are interested in

finding "small" sets that generate a given closed set under a given closure operator. In particular we consider the closure operator induced by a set of implications, and the closure operator induced by a formal context. By saying small sets, we mean two notions of minimality: sets minimal w.r.t set inclusion order, and sets minimal w.r.t cardinality.

Solving the first problem, which is finding the minimal generators and minimum cardinality generators of a set closed under a set of implications, can help the expert during attribute exploration by making the implication questions "simpler". The attribute exploration algorithm asks the minimum number of questions to the expert. This implies that none of the questions asked is redundant. However, it is still possible that a question can be shortened by removing redundant attributes. A similar problem has been considered in the field of relational databases a long time ago. The problem considered there is known as finding *keys* of a relation. In [LO78] it has been shown that checking whether a relation has a key of size less than a specified integer is NP-complete. Here we show that the problem of checking whether a set closed under implications has a generator of cardinality less than a specified size is NP-complete. We also show that determining the number of minimal generators of an implication closed set is #P-complete, i.e., this counting problem is intractable. Moreover, we show that determining the number of minimum cardinality generators is #·coNP-complete, i.e., even harder than counting minimal generators.

The second problem, which is the problem of finding small generators of a set closed under the closure operator induced by a formal context, is known as finding the minimal generators of a concept intent. Different aspects of it have been considered in the literature [NVRG05, FVG05]. Minimal generators of a concept intent play an important rôle in incremental lattice construction, and lattice merge algorithms. In [NVRG05], the behaviour of minimal generators upon additions to the underlying context's attribute set has been investigated, and a method for computing the family of minimal generators has been presented. Here, we show that the problem of checking whether a concept intent has a generator of cardinality less than a specified size is NP-complete. Moreover, we show that counting minimal generators of a concept intent is #P-complete. Actually it is not surprising that the mentioned problems about generators of concept intents and generators of an implication closed set are of the same complexity. In fact, the closure operator induced by a formal context, and the closure operator induced by the set of implications that are valid in this formal context coincide. The proofs of these complexity results are given in Chapter 6.

1.5 Structure of the thesis

- In Chapter 2 we give a brief introduction to DLs. We only introduce the notions we are going to use in the later chapters. We define the syntax and semantics of the DLs we are going to use in the later chapters, define the standard inferences subsumption and instance checking, and the non-standard inference least common subsumer.

- In Chapter 3 we briefly introduce FCA. We first recall preliminaries from order theory, and then give the definitions of the basic notions of FCA. Then we describe attribute exploration, which is going to be our main tool in the later chapters, and give a small example that demonstrates how the exploration algorithm works.

- In Chapter 4 we describe how tools from Formal Concept Analysis can be used to support the bottom-up construction of Description Logic knowledge bases. We address the problem of supporting bottom-up construction in a more general setting, where the user is allowed to re-use concepts from an existing knowledge base that is written in an expressive Description Logic. To this purpose, we introduce a practical method that uses Formal Concept Analysis as a tool. We report the first experimental results of the method. We also recall two other uses of Formal Concept Analysis in the literature in a similar setting, and show that all these approaches are, from a lattice-theoretic point of view, instances of a more general approach.

- In Chapter 5 we present an approach for extending both the terminological and the assertional part of a Description Logic knowledge base by using information provided by the knowledge base and by a domain expert. Our approach provides a basis for formally well-founded techniques and tools that support the knowledge engineer in checking whether a knowledge base contains all the relevant information about the application domain, and to extend the knowledge base appropriately if this is not the case. The use of techniques from Formal Concept Analysis ensures that, on the one hand, the interaction with the expert is kept to a minimum, and, on the other hand, it enables us to show that the extended knowledge base is complete in a certain, well-defined sense. Due to the open-world semantics of Description Logic knowledge bases, we cannot use existing tools of Formal Concept Analysis. For our approach, we introduce our own variant of Formal Concept Analysis that can deal with partial knowledge.

- In Chapter 6 we investigate the problem of finding the generators of a closed set. More precisely, we are interested in finding "small" sets that generate a given closed set under a given closure operator. In

particular we consider the closure operator induced by a set of implications (Section 3.3), and the closure operator induced by a formal context (Section 3.2). By saying small sets, we mean two notions of minimality, namely, sets minimal w.r.t. set inclusion, and sets minimal w.r.t. cardinality. We analyze the computational complexity of some decision and counting problems related to these two problems. We show that for both of these closure operators, the problem of deciding whether a generator with cardinality less than a given integer exists is NP-complete, and the problem of counting minimal generators is #P-complete. We also show that for the first closure operator, the problem of counting generators with minimum cardinality is #·coNP-complete.

- In Chapter 7 we give a summary of our results, and discuss possible future directions and extensions of our work.

Some of the results presented in this work have already been published. The results on supporting bottom-up construction of DL knowledge bases presented in Chapter 4 have appeared in [BST04a, BST04b, BST07]. The improvement of the method mentioned in the same chapter has appeared in [Ser06], and the more general view of the approach, also presented in that chapter, has been published in [BS04]. The results on completing DL knowledge bases presented in Chapter 5 have appeared in [BGSS06, BGSS07a, BGSS07b].

Chapter 2

Description Logics

In the present chapter we introduce Description Logics as a formal language for representing knowledge and reasoning about knowledge. We start with Section 2.1, where we define the syntax of the languages we are going to use in the later chapters. In Section 2.2 we define the semantics of these languages. In Section 2.3 we define the inference problems subsumption, satisfiability, consistency and instance checking, which are also called standard inferences. We conclude the chapter with Section 2.4, where we define the non-standard inference problem of computing the least common subsumer.

2.1 Syntax

In order to define concepts in a DL knowledge base, one starts with a set N_C of unary predicates called *concept names* and a set N_R of binary predicates called *role names*, and builds more complex *concept descriptions* using the constructors provided by the particular description language used. In the present work, we consider the DL $\mathcal{ALC}$ and its sublanguages $\mathcal{ALE}$ and $\mathcal{EL}$, which allow for concept descriptions built from the indicated subsets of the constructors shown in Table 2.1 (for a comprehensive source on DLs, see [BCM$^+$03]). In this table, r stands for a role name, A for a concept name, and C, D for arbitrary concept descriptions. According to this table concept descriptions expressible by $\mathcal{ALC}$ are: bottom-concept ($\bot$), top-concept ($\top$), A (where $A \in N_C$), conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), full negation ($\neg C$), existential restriction ($\exists r.C$) and value restriction ($\forall r.C$). Note that, while $\mathcal{ALC}$ allows for negation of arbitrary concept descriptions, $\mathcal{ALE}$ allows only for *atomic negation*, i.e., negation of concept names, and $\mathcal{EL}$ not even that.

Definition 2.1.1 (Concept definition, GCI). A *concept definition* is an expression of the form $A \equiv C$, where A is a concept name, and C is a complex concept description. It assigns the concept name A to the concept description C. The concept names occurring on the left-hand side of

DL	$\top$	$\bot$	$\neg A$	$\neg C$	$C \sqcap D$	$C \sqcup D$	$\exists r.C$	$\forall r.C$
$\mathcal{EL}$	√				√		√	
$\mathcal{ALE}$	√	√	√		√		√	√
$\mathcal{ALC}$	√	√	√	√	√	√	√	√

Table 2.1: Syntax of $\mathcal{EL}$, $\mathcal{ALE}$ and $\mathcal{ALC}$

a concept definition are called *defined concepts*, and the others are called *primitive concepts*. A *general concept inclusion (GCI)* is an expression of the form $C \sqsubseteq D$, where C and D are two possibly complex concept descriptions. It states a subconcept/superconcept relation between the two concept descriptions. ◇

Typically, a DL knowledge base consists of two components called terminological box and assertional box, or TBox and ABox for short.

Definition 2.1.2 (TBox, ABox). A *TBox* is a finite unambiguous set of concept definitions, i.e., each name has at most one definition. It is called an *acyclic TBox* if none of the concept definitions inside the TBox refers directly or indirectly to the name it defines. Otherwise, it is called a *cyclic TBox*. A finite set of GCIs is called a *general TBox*.

An *ABox* is a finite set of concept and role assertions. A *concept assertion* is an expression of the form $C(a)$ where C is a concept description and a is an individual name. A *role assertion* is an expression of the form $r(a, b)$ where r is a role name, and a and b are individual names. We represent a knowledge base as $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. ◇

2.2 Semantics

The semantics of DL concept descriptions is defined in terms of *interpretations*, which are defined as follows:

Definition 2.2.1 (Interpretation). An interpretation is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is called the *domain*, and $\cdot^{\mathcal{I}}$ is called the *interpretation function* of $\mathcal{I}$. The domain is a non-empty set and the interpretation function maps each concept name $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is inductively defined as shown in the upper part of Table 2.2. ◇

Definition 2.2.2 (Model). We say that an interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ iff it satisfies all concept definitions in $\mathcal{T}$, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \equiv C$ in $\mathcal{T}$. It is a model of a general TBox $\mathcal{T}$ iff it satisfies all concept inclusions in $\mathcal{T}$, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all $C \sqsubseteq D$ in $\mathcal{T}$.

Constructor name	Syntax	Semantics
top-concept	$\top$	$\Delta^{\mathcal{I}}$
bottom-concept	$\bot$	$\emptyset$
atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$r(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Table 2.2: Syntax and semantics of commonly used constructors

Similarly, we say that an interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ iff it satisfies all concept and roles assertions, i.e., for every $C(a)$ in $\mathcal{A}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds and for every $r(a,b)$ in $\mathcal{A}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ holds. $\mathcal{I}$ is said to be a model of the knowledge base $(\mathcal{T}, \mathcal{A})$ iff it is a model of both $\mathcal{T}$ and $\mathcal{A}$. $\diamond$

2.3 Standard inferences

DLs are not only used for storing concept definitions, and concept or role assertions, but also for reasoning about them, i.e., deducing implicit knowledge from the explicitly represented knowledge. One of the most traditional inference problems in DLs is the problem of deciding subconcept/superconcept relationships between concept descriptions, namely subsumption.

Definition 2.3.1 (Subsumption). We say that a concept description D *subsumes* the concept description C (or equivalently C is *subsumed* by D) w.r.t. the TBox $\mathcal{T}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models of $\mathcal{T}$. We write it as $C \sqsubseteq_{\mathcal{T}} D$. If C is subsumed by D w.r.t. the empty TBox, we just write $C \sqsubseteq D$. Two concept descriptions are called *equivalent* w.r.t. $\mathcal{T}$ (written $C \equiv_{\mathcal{T}} D$) iff they subsume each other w.r.t. $\mathcal{T}$, i.e., $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$. $\diamond$

The subsumption relation $\sqsubseteq_{\mathcal{T}}$ is a quasi order (i.e., reflexive and transitive), but in general not a partial order since it need not be antisymmetric, i.e., there may exist equivalent descriptions that are not syntactically equal. As usual, the quasi order $\sqsubseteq_{\mathcal{T}}$ induces a partial order $\sqsubseteq_{\mathcal{T}}^{\equiv}$ on the equivalence classes of concept descriptions:

$$[C]_{\equiv} \sqsubseteq_{\mathcal{T}}^{\equiv} [D]_{\equiv} \text{ iff } C \sqsubseteq_{\mathcal{T}} D,$$

where $[C]_{\equiv} := \{E \mid C \equiv_{\mathcal{T}} E\}$ is the equivalence class of C, and $[D]_{\equiv}$ is defined respectively. When talking about the *subsumption hierarchy* of a set of concept descriptions, we mean this induced partial order.

It should be noted that subsumption w.r.t. acyclic TBoxes can be reduced to subsumption of concept descriptions by *expanding the TBox*, i.e. by replacing the defined concepts by their definitions until no more defined concepts occur in the concept descriptions to be tested for subsumption. However, this reduction cannot be used to obtain the complexity results for subsumption w.r.t. acyclic TBoxes mentioned above since the expansion process may cause an exponential blow-up of the concept descriptions [Neb90].

In addition to subsumption, there are other important standard inferences, which are formally defined as follows:

Definition 2.3.2 (Satisfiability, consistency, instance). We say that a concept description C is *satisfiable* w.r.t. a TBox $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. We say that a knowledge base $(\mathcal{T}, \mathcal{A})$ is *consistent* iff $\mathcal{T}$ and $\mathcal{A}$ have a common model. We say that an individual a is an *instance* of a concept description C w.r.t. $\mathcal{T}$ and $\mathcal{A}$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for every common model $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$. We write it as $\mathcal{T}, \mathcal{A} \models C(a)$. $\diamond$

The inferences mentioned above, namely subsumption, satisfiability, consistency and instance are also called *standard inferences*. In a DL that allows for conjunction and full negation, the subsumption and the satisfiability problems are inter-reducible in polynomial time, and the same is true for the instance and the consistency problems. In addition, the satisfiability problem can always be reduced in polynomial time to the consistency problem. For concept descriptions C, D and individual name a, these reductions are shown below:

- *subsumption to (un)satisfiability*: C is subsumed by D w.r.t. $\mathcal{T}$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$.
- *(un)satisfiability to subsumption*: C is unsatisfiable w.r.t. $\mathcal{T}$ iff C is subsumed by $\bot$ w.r.t. $\mathcal{T}$.
- *instance to (in)consistency*: a is an instance of C w.r.t. $\mathcal{T}$ and $\mathcal{A}$ iff $(\mathcal{T}, \mathcal{A} \cup \{\neg C(a)\})$ is inconsistent.
- *(in)consistency to instance*: $(\mathcal{T}, \mathcal{A})$ is inconsistent iff $\mathcal{T}, \mathcal{A} \models C(a)$ and $\mathcal{T}, \mathcal{A} \models \neg C(a)$ where C and a are arbitrarily chosen.
- *satisfiability to consistency*: C is satisfiable w.r.t. $\mathcal{T}$ iff $(\mathcal{T}, \{C(a)\})$ is consistent, where a is arbitrarily chosen.

The complexity of the problems mentioned above depends on the DL under consideration, and on what kind of TBox formalism is used. For the DL

$\mathcal{ALC}$, these problems are PSPACE-complete if the TBox is empty [SSS91], or if it is acyclic [Lut99, Lut02]. It becomes EXPTIME-complete in the presence of general TBoxes [Sch91]. For $\mathcal{ALE}$, subsumption w.r.t. the empty TBox is NP-complete [DLN+92], and it is EXPTIME-complete w.r.t. general TBoxes [BBL05a, BBL05b]. For the DL $\mathcal{EL}$, subsumption is polynomial w.r.t. the empty TBox [BKM99], and w.r.t. (a)cyclic TBoxes [Baa03d]. Later on in [Bra04] it has been shown that it stays polynomial even in the presence of general TBoxes. In his Ph.D thesis [Bra06], Brandt has investigated the borders of tractability for various extensions of $\mathcal{EL}$.

Investigating the trade-off between the expressivity of DLs and the complexity of their inference problems is one of the most important issues of DL research. As a consequence of this, the complexity of reasoning in various DLs of different expressive power is now well-known (see [Don03] for an overview of these complexity results). In addition, there are now highly optimized DL reasoners like FaCT [Hor98] (its successor FaCT++ [TH06]), RACER [HM01b] (its successor RACERPRO), Pellet [SP04] and KAON2 [Mot06], which—despite their high worst-case complexity—behave very well in practice for DLs that are considerably more expressive than $\mathcal{ALC}$. For a comprehensive list of DL reasoners, see the web page[1] maintained by Sattler.

2.4 Non-standard inferences

In addition to the standard inferences mentioned in the previous section, so-called *non-standard inferences* have been introduced and investigated in the DL community. In the present work, we are only interested in the non-standard inference problem of computing the *least common subsumer (lcs)*. For an overview on non-standard inferences, see [Küs01, Bra06]. In the following definition, let $\mathcal{L}$ be some DL.

Definition 2.4.1 (Least common subsumer). Given a collection $C_1, \ldots, C_n$ of $\mathcal{L}$-concept descriptions, the *least common subsumer* (lcs) of $C_1, \ldots, C_n$ in $\mathcal{L}$ is the most specific $\mathcal{L}$-concept description that subsumes $C_1, \ldots, C_n$, i.e., it is an $\mathcal{L}$-concept description D such that

1. $C_i \sqsubseteq D$ for each $i = 1, \ldots, n$ *(D is a common subsumer);*
2. if E is an $\mathcal{L}$-concept description satisfying $C_i \sqsubseteq E$ for each $i = 1, \ldots, n$, then $D \sqsubseteq E$ *(D is the smallest such concept description).*

◇

The problem of computing the lcs was originally introduced for concept descriptions, i.e., w.r.t. the empty TBox. In the presence of acyclic TBoxes, one can first expand the concept descriptions in question, and apply this

[1] `http://www.cs.man.ac.uk/~sattler/reasoners.html`

inference. As an easy consequence of Definition 2.4.1, when it exists, then lcs is unique up to equivalence, which justifies talking about *the* lcs. In addition, the n-ary lcs as defined above can be reduced to the binary lcs (the case $n = 2$ above). Indeed, it is easy to see that the lcs of $C_1, \ldots, C_n$ can be obtained by building the lcs of C_1, C_2, then the lcs of this concept description with C_3, etc. Thus, it is enough to devise algorithms for computing the binary lcs.

It should be noted, however, that the lcs need not always exist. This can have different reasons: (a) there may not exist a concept description in $\mathcal{L}$ satisfying item (1) of the definition (i.e., subsuming $C_1, \ldots, C_n$); (b) there may be several subsumption incomparable minimal concept descriptions satisfying (1) of the definition; (c) there may be an infinite chain of more and more specific descriptions satisfying (1) of the definition. Obviously, (a) cannot occur for DLs containing the top concept. It is easy to see that, for DLs allowing for conjunction of descriptions, (b) cannot occur.

It is also clear that in DLs allowing for disjunction, the lcs of $C_1, \ldots, C_n$ is their disjunction $C_1 \sqcup \ldots \sqcup C_n$. In this case, the lcs is not really of interest. Instead of extracting properties common to $C_1, \ldots, C_n$, it just gives their disjunction, which does not provide us with new information. This means, it makes sense to look at the lcs in $\mathcal{EL}$ and $\mathcal{ALE}$, but not in $\mathcal{ALC}$. Both for $\mathcal{EL}$ and $\mathcal{ALE}$, the lcs always exists, and can be effectively computed [BKM99]. For $\mathcal{EL}$, the size and computation time for the binary lcs is polynomial, but exponential in the n-ary case. For $\mathcal{ALE}$, already the size of the binary lcs may be exponential in the size of the input concept descriptions.

The SONIC[2] system [TK04a, TK04b] implements an lcs algorithm, as well as some other non-standard inferences. It can be plugged into the ontology editors Protégé [KFNM04] and OilEd [BHGS01], and accesses a DL reasoner like FaCT or RACER over the DIG[3] interface.

[2]Simple Ontology Non-standard Inference Component
[3]Description Logic Implementation Group

Chapter 3

Formal Concept Analysis

In the present chapter we introduce Formal Concept Analysis. We start with Section 3.1, where we recall some notions from order theory. In Section 3.2 we introduce the basic notions of Formal Concept Analysis, namely formal context, formal concept, and concept lattice. In Section 3.3 we introduce implications between the attributes of a formal context, and a method to compute a canonical base of all implications that hold in a formal context, namely the stem base. In Section 3.4 we introduce attribute exploration, which is the interactive knowledge acquisition method of Formal Concept Analysis. In the coming chapters, it is going to be our main tool for supporting bottom-up construction and completion of Description Logic knowledge bases.

3.1 Order-theoretic preliminaries

We start with introducing order-theoretic notions that we are going to use in the coming sections of this chapter.

Definition 3.1.1 (Partial order). A binary relation $\leq$ on a set M is called a *partial order* if it satisfies the following conditions for all elements $x, y, z \in M$:

1. $x \leq x$,
2. $x \leq y$ and $y \leq x$ imply $x = y$,
3. $x \leq y$ and $y \leq z$ imply $x \leq z$.

These conditions are referred to, respectively as *reflexivity*, *antisymmetry* and *transitivity*. A set M equipped with a partial order relation $\leq$ is said to be a *partially ordered set*, or *poset* for short. A relation $\leq$ on a set M which is reflexive and transitive but not necessarily antisymmetric is called a *quasi-order*. ◇

Definition 3.1.2 (Infimum, supremum). Let $(M, \leq)$ be a poset and A a subset of M. A *lower bound* of A is an element s of M with $s \leq a$ for all $a \in A$. An *upper bound* of A is defined dually. If it exists, the largest element in the set of all lower bounds of A is called the *infimum* of A and it is denoted by $inf\ A$ or $\wedge A$; dually a least upper bound is called *supremum* and denoted by *sup* A or $\vee A$. Infimum and supremum are often also called *meet* and *join*. ◇

Definition 3.1.3 (Lattice, complete lattice). A partially ordered set $\mathcal{V} = (V, \leq)$ is called a *lattice* if, for any two elements in V, the infimum and the supremum always exist. $\mathcal{V}$ is called a *complete lattice* if, for any subset of V, the supremum and the infimum exist. Every complete lattice $\mathcal{V}$ has a largest element $\vee V$, called the *unit element*, denoted by 1_V. Dually, the smallest element 0_V is called the *zero element*. ◇

The definition of a complete lattice assumes that supremum and infimum exist for every $X \subseteq V$. For $X = \emptyset$, we have $\wedge\emptyset = 1_V$, and $\vee\emptyset = 0_V$, from which follows that $V \neq \emptyset$ for every complete lattice. Every non-empty finite lattice is a complete lattice.

Definition 3.1.4 (Closure operator). Let S be a set and φ a mapping from the power set of S into the power set of S. Then φ is called a *closure operator* on S if it is

- *extensive:* $A \subseteq \varphi(A)$ for all $A \subseteq S$;
- *monotone:* $A \subseteq B$ implies $\varphi(A) \subseteq \varphi(B)$ for all $A, B \subseteq S$; and
- *idempotent:* $\varphi(\varphi(A)) = \varphi(A)$.

We say that a set $A \subseteq S$ is *φ-closed* if $A = \varphi(A)$. The set of all φ-closed sets, i.e. $\{A \subseteq S \mid A = \varphi(A)\}$ is called a *closure system*. ◇

3.2 Formal contexts and formal concepts

Formal Concept Analysis (FCA) [GW99] is a field of applied mathematics that is based on a lattice-theoretic formalization of the notions of a concept and of a hierarchy of concepts. It aims to facilitate the use of mathematical reasoning for conceptual data analysis and knowledge processing.

In FCA, one represents data in form of a *formal context*, which in its simplest form is a way of specifying which attributes are satisfied by which objects. Formally, a formal context is defined as follows:

Definition 3.2.1 (Formal context). A *formal context* is a triple $\mathbb{K} = (G, M, I)$, where G is a set of objects, M is a set of attributes, and $I \subseteq G \times M$ is a relation that associates each object g with the attributes satisfied by g. In order to express that an object g is in relation I with an attribute m, we write gIm. ◇

A formal context is usually visualized as a cross table, where the rows represent the objects, and the columns represent the attributes of the context. A cross in column m of row g means that object g has attribute m, and the absence of a cross means that g does not have attribute m. In the present work we consider formal contexts with finite attribute sets.

Definition 3.2.2 (Derivation operator). Let $\mathbb{K} = (G, M, I)$ be a formal context. For a set of objects $A \subseteq G$, we define the set of attributes that are satisfied by all objects in A as follows:

$$A' := \{m \in M \mid \forall g \in A.\ gIm\}.$$

Similarly, for a set of attributes $B \subseteq M$, we define the set of objects that satisfy all attributes in B as follows:

$$B' := \{g \in G \mid \forall m \in B.\ gIm\}.$$

◇

For $A_1 \subseteq A_2 \subseteq G$ (resp. $B_1 \subseteq B_2 \subseteq M$), it is easy to see that

- $A_2' \subseteq A_1'$ (resp. $B_2' \subseteq B_1'$),
- $A_1 \subseteq A_1''$ and $A_1' = A_1'''$ (resp. $B_1 \subseteq B_1''$ and $B_1' = B_1'''$).

As an easy consequence one obtains that the $(\cdot)''$ operation is a *closure operator* on both G and M.

Definition 3.2.3 (Formal concept). Let $\mathbb{K} = (G, M, I)$ be a formal context. A *formal concept* of $\mathbb{K}$ is a pair (A, B) where $A \subseteq G$, $B \subseteq M$ such that $A' = B$ and $B' = A$. We call A the *extent*, and B the *intent* of (A, B). If (A_1, B_1) and (A_2, B_2) are two formal concepts of a context, and $A_1 \subseteq A_2$ (or, equivalently $B_2 \subseteq B_1$), we say that (A_1, B_1) is a *subconcept* of (A_2, B_2), and write $(A_1, B_1) \leq (A_2, B_2)$. The relation $\leq$ is called the *hierarchical order* of formal concepts. ◇

The set of all formal concepts of a context $\mathbb{K} = (G, M, I)$ ordered with the hierarchical order form a complete lattice, called the *concept lattice* of $\mathbb{K}$ and it is denoted by $\underline{\mathfrak{B}}(G, M, I)$. The Basic Theorem on Concept Lattices [GW99] shows that a concept lattice is a complete lattice, and gives the definition of infimum and supremum in a concept lattice.

Theorem 3.2.4 (The Basic Theorem on Concept Lattices). *The concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice in which infimum and supremum are given by the following equations:*

$$\begin{aligned} \textstyle\bigwedge_{t\in T}(A_t, B_t) &= \left(\textstyle\bigcap_{t\in T} A_t, \left(\textstyle\bigcup_{t\in T} B_t\right)''\right), \\ \textstyle\bigvee_{t\in T}(A_t, B_t) &= \left(\left(\textstyle\bigcup_{t\in T} A_t\right)'', \textstyle\bigcap_{t\in T} B_t\right). \end{aligned}$$

The following is an easy consequence of the definition of formal concepts and the properties of the $(\cdot)'$ operator:

Lemma 3.2.5. *All formal concepts are of the form (A'', A') for a subset A of G, and any such pair is a formal concept.*

The dual of this lemma is also true, i.e., all formal concepts are of the form (B', B'') for a subset B of M, and any such pair is a formal concept.

Given a formal context, the first step for analyzing this context is usually to compute the concept lattice. If the context is finite, then Lemma 3.2.5 implies that the concept lattice can, in principle, be computed by enumerating the subsets A of G, and applying the operators $(\cdot)'$ and $(\cdot)''$. However, this naïve algorithm is usually very inefficient. It is well known that the set of all formal concepts of a context can be exponential in the size of the considered context (i.e., $|G| \times |M|$), e.g., when the resulting concept lattice is a Boolean lattice. In [Kuz01], it was shown that the problem of determining the size of a concept lattice is #P-complete (see Section 6.1 for counting complexity classes), i.e., even determining the number of concepts of a context is intractable. There are several polynomial-delay algorithms for computing the set of all concepts [Nor78, Gan84, Bor86, Kuz93, GMA95, NR99, STB$^+$00, VM01]. For a detailed analysis and evaluation of these algorithms, see [KO02].

In the present work, we are going to use the most popular of the above-mentioned algorithms, namely Ganter's *next closure* algorithm [Gan84] as the main tool from FCA. The algorithm is not specifically tailored for computing the set of all concepts of a context. It computes all closed sets of a given closure operator. Given the closure operator $(\cdot)''$ on the attribute (object) set of a formal context $\mathbb{K}$, it computes all concept intents (resp. extents) of $\mathbb{K}$ in a certain lexicographic order, called the lectic order.

Definition 3.2.6 (Lectic order). Assume that $M = \{m_1, \ldots, m_n\}$ and fix some linear order $m_1 < m_2 < \cdots < m_n$ on M. This order imposes a linear order on the power set of M, called the *lectic order*, which we also denote by $<$. For $m_i \in M$ and $A, B \subseteq M$ we define:

$$A <_i B \quad \text{iff} \quad m_i \in B,\ m_i \notin A \text{ and } \forall j < i.\ (m_j \in A \Leftrightarrow m_j \in B).$$

The order $<$ is the union of these orders $<_i$, i.e.,

$$A < B \quad \text{iff} \quad A <_i B \text{ for some } i \in M.$$

$\diamond$

Obviously, $<$ extends the strict subset order, and thus $\emptyset$ is the smallest and M is the largest set w.r.t. $<$. The following proposition shows how one can find the smallest closed set lectically greater than a given set.

Proposition 3.2.7 (Next closure). *Given a closure operator φ on M and a set $A \subsetneq M$, the smallest φ-closed set greater than A w.r.t. the lectic order is*

$$\varphi((A \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$$

where j is maximal such that $A <_j \varphi((A \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$.

In order to find all closures of φ, one starts with the lectically smallest φ-closed set $\varphi(\emptyset)$, and applies the proposition successively until the lectically largest φ-closed set M is reached.

3.3 Implications between attributes

Given a formal context, one other common method to analyze it is to find (a canonical base of) the implications between the attributes of this context. Implications between attributes are constraints that hold in a given context. They are statements of the form

> "Every object that satisfies the attributes $m_{i1}, \ldots, m_{ik}$ also satisfies the attributes $m_{j1}, \ldots, m_{j\ell}$."

Formally, an implication between attributes is defined as follows:

Definition 3.3.1 (Implication between attributes). Let $\mathbb{K} = (G, M, I)$ be a formal 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 of $\mathbb{K}$ that has all of the attributes in L also has all of the attributes in R, i.e., if $L' \subseteq R'$. We denote the set of all implications that hold in $\mathbb{K}$ by $Imp(\mathbb{K})$, and call it the *implicational theory* of $\mathbb{K}$. ◇

Example 3.3.2. Consider $\mathbb{K}_{countries}$ given in Example 1.2.1. In this context, the implication $\{nw, SCp\} \to \{G8\}$ holds since the countries that have nuclear weapons, and that are permanent members of the UN Security Council, namely USA, France and UK, are also members of G8. However the implication $\{G8\} \to \{nw, SCp\}$ does not hold since Italy, which is a G8 member, neither has nuclear weapons, nor is a permanent member of the UN Security Council. ◇

It is easy to see that an implication $L \to R$ holds in $\mathbb{K}$ iff R is contained in the $(\cdot)''$-closure of L, i.e., if $R \subseteq L''$. A set of implications induces its own closure operator.

Definition 3.3.3 (Implicational closure). Let $\mathcal{L}$ be a set of implications. For a set $P \subseteq M$, the *implicational closure* of P with respect to $\mathcal{L}$, denoted by $\mathcal{L}(P)$, is the smallest subset Q of M such that

- $P \subseteq Q$, and

- $L_i \to R_i \in \mathcal{L}$ and $L_i \subseteq Q$ imply $R_i \subseteq Q$.

It is easy to see that $\mathcal{L}(\cdot)$ is indeed a closure operator. ◇

From the point of view of logic, computing the implicational closure of a set of attributes P is just computing consequences in propositional Horn logic. In fact, the notions we have just defined can easily be reformulated in propositional logic. To this purpose, we view the attributes as propositional variables. An implication $L \to R$ can then be expressed by the formula $\phi_{L\to R} := \bigwedge_{r\in R}(\bigwedge_{\ell\in L} \ell \to r)$. Let $\Gamma_{\mathcal{L}}$ be the set of formulae corresponding to the set of implications $\mathcal{L}$. Then

$$\mathcal{L}(P) = \{b \in M \mid \Gamma_{\mathcal{L}} \cup \{\bigwedge_{p\in P} p\} \models b\},$$

where $\models$ stands for classical propositional consequence. Obviously, the formulae in $\Gamma_{\mathcal{L}}$ are Horn clauses. For this reason, the implication closure $\mathcal{L}(B)$ of a set of attributes B can be computed in time linear in the size of $\mathcal{L}$ and B using methods for deciding satisfiability of sets of propositional Horn clauses [DG84]. Alternatively, these formulae can be viewed as expressing functional dependencies in relational database, and thus the linearity result can also be obtained by using methods for deriving new functional dependencies from the given ones [Mai83].

Definition 3.3.4 (Implication base). The implication $L \to R$ is said to *follow* from a set $\mathcal{J}$ of implications if $R \subseteq \mathcal{J}(L)$. The set of implications $\mathcal{J}$ is called *complete* for a set of implications $\mathcal{L}$ if every implication in $\mathcal{L}$ follows from $\mathcal{J}$. It is called *sound* for $\mathcal{L}$ if every implication that follows from $\mathcal{J}$ is contained in $\mathcal{L}$. A set of implications $\mathcal{J}$ is called a *base* for a set of implications $\mathcal{L}$ if it is both sound and complete for $\mathcal{L}$, and no strict subset of $\mathcal{J}$ satisfies this property. ◇

Again, the consequence operation between implications coincides with the usual logical notion of consequence if one translates implications into Horn clauses, as described above.

If $\mathcal{J}$ is sound and complete for $Imp(\mathbb{K})$, then the two closure operators that we have introduced until now coincide, i.e., $B'' = \mathcal{J}(B)$ for all $B \subseteq M$. Consequently, given a base $\mathcal{J}$ for $Imp(\mathbb{K})$, any question of the form "$B_1 \to B_2 \in Imp(\mathbb{K})$?" can be answered in time linear in the size of $\mathcal{J} \cup \{B_1 \to B_2\}$ since it is equivalent to asking whether $B_2 \subseteq B_1'' = \mathcal{J}(B_1)$.

The implicational theory $Imp(\mathbb{K})$ of a formal context $\mathbb{K}$ can be large. Thus, one is interested in small bases generating $Imp(\mathbb{K})$. There may exist different bases of $Imp(\mathbb{K})$, and not all of them need to be of minimum cardinality. A base $\mathcal{J}$ of $Imp(\mathbb{K})$ is called *minimum base* iff no base of $Imp(\mathbb{K})$ has a cardinality smaller than the cardinality of $\mathcal{J}$. Duquenne and Guigues have given a description of such a minimum base [GD86] for formal contexts

with a finite set of attributes. It is called the *Duquenne-Guigues Base* or the *stem base* of a formal context.[1] The description is based on the notion of pseudo-intent.

Definition 3.3.5 (Pseudo-intent). A set $P \subseteq M$ is called a *pseudo-intent* of $\mathbb{K} = (G, M, I)$ if $P \neq P''$ and $Q'' \subseteq P$ holds for all pseudo-intents $Q'' \subsetneq P$. $\diamond$

The Duquenne-Guigues Base of a formal context $\mathbb{K}$ consists of the implications that have the pseudo-intents of $\mathbb{K}$ as left-hand sides.

Theorem 3.3.6 (Duquenne-Guigues Base). *The set of implications*

$$\mathcal{L} := \{P \to P'' \mid P \text{ a pseudo-intent of } \mathbb{K}\}$$

is a minimum base of Imp$(\mathbb{K})$*.*

Example 3.3.7. The Duquenne-Guigues Base of $\mathbb{K}_{countries}$ given in Example 1.2.1 is the set of implications

$$\mathcal{L}_{countries} = \left\{ \begin{array}{rcl} \{\} & \to & \{UN\} \\ \{nw, UN\} & \to & \{SCp, G8\} \\ \{SCp, UN\} & \to & \{nw, G8\} \\ \{SCt, G8, UN\} & \to & \{EU\} \\ \{EU, UN\} & \to & \{G8\} \end{array} \right\}$$

The implication $\{nw, SCp\} \to \{G8\}$ we have mentioned in Example 3.3.2 follows from $\mathcal{L}_{countries}$, since the closure $\mathcal{L}_{countries}(\{nw, SCp\}) = \{nw, UN, SCp, G8\}$ contains the conclusion of the implication. $\diamond$

The recursive definition of pseudo-intents does not yield a practical algorithm for generating them. The following property enables us to find the pseudo-intents of a formal context.

Lemma 3.3.8 (Closure system of intents and pseudo-intents). *The set of all intents and pseudo-intents of a formal context is a closure system.*

By using the next closure algorithm, we can compute the closed sets of this closure system, i.e., the intents and the pseudo-intents of the underlying formal context. We start with the lectically smallest intent or pseudo-intent, which is the closure of $\emptyset$, and apply Proposition 3.2.7 successively until we reach the lectically largest intent M. What we need is the closure operator inducing the closure system of intents and pseudo-intents. It is obtained by a modification of the implicational closure operator $\mathcal{L}(\cdot)$ given in Definition 3.3.3.

[1] From now on we will simply say Duquenne-Guigues Base of a formal context instead of saying Duquenne-Guigues Base of the implicational theory of a formal context.

Algorithm 1 Duquenne-Guigues Base computation

1: **Initialization**
2: $\mathbb{K}$ {the input formal context}
3: $\mathcal{L}_0 := \emptyset$ {initial empty set of implications}
4: $P_0 := \emptyset$ {lectically smallest $\mathcal{L}_0$-closed subset of M}
5: $i := 0$
6: **while** $P_i \neq M$ **do**
7: Compute P_i'' w.r.t. $\mathbb{K}$
8: **if** $P_i \neq P_i''$ **then** {P_i is a pseudo-intent}
9: $\mathcal{L}_{i+1} := \mathcal{L}_i \cup \{P_i \to P_i'' \setminus P_i\}$
10: **end if**
11: $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P_i <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
12: $i := i + 1$
13: **end while**

Definition 3.3.9 (Implication pseudo-closure). For a set $P \subseteq M$, the *implication pseudo-closure* of P with respect to a set of implications $\mathcal{L}$, denoted by $\mathcal{L}^*(P)$, is the smallest subset Q of M such that

- $P \subseteq Q$, and
- $L_i \to R_i \in \mathcal{L}$ and $L_i \subsetneq Q$ imply $R_i \subseteq Q$.

$\diamond$

Note that we actually need implication pseudo-closure for computing the Duquenne-Guigues Base of a formal context $\mathbb{K}$, but the definition of implication pseudo-closure itself refers to the set of valid implications of $\mathbb{K}$. This does not cause a problem since we compute the intents and the pseudo-intents in the lectic order. The lectic order makes sure that it is sufficient to use the already computed part of the Duquenne-Guigues Base when computing the pseudo-closure. Given $\mathcal{L}$, the pseudo-closure of a set $B \subseteq M$ can again be computed in time linear in the size of $\mathcal{L}$ and B (e.g., by adapting the algorithms in [DG84, Mai83] appropriately).

Based on these considerations, Duquenne-Guigues Base computation is described in Algorithm 1. One may wonder, why in step 11 we compute the usual implication closure $\mathcal{L}_{i+1}$ rather than the implication pseudo-closure $\mathcal{L}_{i+1}^*$ from Definition 3.3.9. This is due to the fact that the algorithm enumerates the closed and pseudo-closed sets in the lectic order. Since the lectic order extends the strict subset order, in practice it does not matter whether usual implication closure or implication pseudo-closure is used.

Note that while computing the pseudo-intents, we are enumerating all concept intents as well, of which there may be exponentially many in the size of the input context. Unfortunately, currently we do not have a more efficient

algorithm that can enumerate pseudo-intents with polynomial-delay. It is well known that the number of pseudo-intents of a formal context can be exponential in the size of the attribute set, i.e., $|M|$. However, in this case $|G|$ as well as $|I|$ are also exponential in $|M|$. For a long time, it was not known whether the number of pseudo-intents can also be exponential in $|G| \times |M|$. In [Kuz04], it was shown that this can be the case. There, it was also shown that determining the number of pseudo-intents is #P-hard. Another interesting problem about pseudo-intents is how difficult it is to decide whether a given set is a pseudo-intent. In [KO06] it was shown that this problem is in *co*NP. However, a lower bound for the complexity of this problem is still unknown.

3.4 Attribute exploration

In the previous section, we have seen how to compute the Duquenne-Guigues Base of a given formal context. What if the formal context is not available, or it is only partly available?

In many applications, one needs to classify a large (or even infinite) set of objects with respect to a relatively small set of attributes. Moreover, it is often the case that the formal context is not given explicitly as a cross table, but it is only "known" to a domain expert. In such cases, Ganter's interactive *attribute exploration* algorithm [Gan84] has proved to be a useful method to compute the Duquenne-Guigues Base and efficiently capture the expert's knowledge. Assume we have the following setting. There is an application domain represented as a formal context $\mathbb{K}$, and a domain expert that knows only part of $\mathbb{K}$. However, (s)he is able to answer if an implication holds in $\mathbb{K}$ and, in case it does not hold he is able to give a counterexample. By asking implication questions to the domain expert, the attribute exploration method computes a base for $Imp(\mathbb{K})$ and a subcontext $\mathbb{K}'$ of $\mathbb{K}$ such that $Imp(\mathbb{K}') = Imp(\mathbb{K})$. For each implication question, the expert either says that it holds in $\mathbb{K}$, in which case the implication is added to the base, or he gives a counterexample from $\mathbb{K}$, which is then added to $\mathbb{K}'$.

The following proposition shows that the method for computing the Duquenne-Guigues Base of a context $\mathbb{K}$ from the previous section, works correctly even if $\mathbb{K}$ is extended during the computation.

Proposition 3.4.1. *Let $\mathbb{K}$ be a formal context and let $P_1, P_2, \ldots, P_n$ be the lectically first n pseudo-intents of $\mathbb{K}$. If $\mathbb{K}$ is extended to $\mathbb{K}'$ such that all the previously accepted implications $P_i \to P_i''$ for $i \in \{1, \ldots, n\}$ still hold in $\mathbb{K}'$, then $P_1, P_2, \ldots, P_n$ are also the lectically first n pseudo-intents of $\mathbb{K}'$.*

Based on these considerations, the attribute exploration algorithm is described in Algorithm 2.

Algorithm 2 Attribute exploration

Initialization
$\mathbb{K}_0$ {initial formal context, possibly empty set of objects}
$\mathcal{L}_0 := \emptyset$ {initial empty set of implications}
$P_0 := \emptyset$ {lectically smallest $\mathcal{L}_0$-closed subset of M}
$i := 0$
while $P_i \neq M$ **do**
  Compute P_i'' w.r.t. $\mathbb{K}_i$
  if $P_i \neq P_i''$ **then**
    Ask the expert if $P_i \to P_i''$ holds
    if yes **then**
      $\mathbb{K}_{i+1} := \mathbb{K}_i$
      $\mathcal{L}_{i+1} := \mathcal{L}_i \cup \{P_i \to P_i'' \setminus P_i\}$
      $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P_i <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
    else
      Get an object o from the expert s.t: $P_i \subseteq o'$ and $P_i'' \not\subseteq o'$
      $\mathbb{K}_{i+1} := \mathbb{K}_i \cup \{o\}$
      $P_{i+1} := P_i$
      $\mathcal{L}_{i+1} := \mathcal{L}_i$
    end if
  else {P_i is an intent, compute the lectically next one}
    $\mathbb{K}_{i+1} := \mathbb{K}_i$
    $\mathcal{L}_{i+1} := \mathcal{L}_i$
    $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P_i <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
  end if
  $i := i + 1$
end while

Example 3.4.2. A closer look at the Duquenne-Guigues Base $\mathcal{L}_{countries}$ in Example 3.3.7, reveals that some of the implications do not actually reflect the case in the real life. For example, the implication $\{\} \to \{UN\}$ says that every country is a UN member. In the context $\mathbb{K}_{countries}$ this holds, however in real life it does not hold. There are countries that are not members of the UN, for instance State of Palestine. Similarly, the implication $\{SCp, UN\} \to \{nw, G8\}$ does not hold in real life either, because China is a member of the UN and it is a permanent member of the UN Security Council, however it is not a member of G8. Table 3.1 shows step by step how Algorithm 2 helps us to complete $\mathbb{K}_{countries}$ such that, at the end, it reflects the state of the world at the time this example has been prepared. At the end of the exploration, the formal context $\mathbb{K}'_{countries}$ looks like in Table 3.2. The concept lattice of the completed context $\mathbb{K}'_{countries}$ is given in Figure 3.1. ◇

$\mathbf{P_i}$	Question	holds?	Action
$\{\}$	$P_i \to \{UN\}$?	no	new obj: State of Palestine
$\{UN\}$	-	-	next P_i
$\{EU\}$	$P_i \to \{G8, UN\}$	no	new obj: Spain
$\{EU\}$	$P_i \to \{UN\}$	yes	accept implication
$\{EU, UN\}$	-	-	next P_i
$\{G8\}$	$P_i \to \{UN\}$	yes	accept implication
$\{G8, UN\}$	-	-	next P_i
$\{G8, EU, UN\}$	-	-	next P_i
$\{SCt\}$	$P_i \to \{UN\}$	yes	accept implication
$\{SCt, UN\}$	-	-	next P_i
$\{SCt, EU, UN\}$	$P_i \to \{G8\}$	no	new obj: Belgium
$\{SCt, G8, UN\}$	$P_i \to \{EU\}$	yes	accept implication
$\{SCt, G8, EU, UN\}$	-	-	next P_i
$\{SCp\}$	$P_i \to \{nw, G8, UN\}$	no	new obj: China
$\{SCp\}$	$P_i \to \{nw, UN\}$	yes	accept implication
$\{nw\}$	$P_i \to \{SCp, UN\}$	yes	accept implication
$\{nw, SCp, UN\}$	-	-	next P_i
$\{nw, SCp, EU, UN\}$	$P_i \to \{G8\}$	yes	accept implication
$\{nw, SCp, G8, UN\}$	-	-	next P_i
$\{nw, SCp, G8, EU, UN\}$	-	-	next P_i
$\{nw, SCp, SCt, UN\}$	$P_i \to \{G8, EU\}$	yes	accept implication
$\{nw, SCp, SCt, G8, EU, UN\}$	-	-	-

Table 3.1: Execution steps of Algorithm 2 on $\mathbb{K}_{countries}$.

$\mathbb{K}'_{countries}$	has nuclear weapons	UN Security Council permanent	UN Security Council temporary	G8 mem.	EU mem.	UN mem.
USA	X	X		X		X
Germany				X	X	X
France	X	X		X	X	X
UK	X	X		X	X	X
Turkey						X
Qatar			X			X
Italy			X	X	X	X
State of Palestine						
China	X	X				X
Spain					X	X
Belgium			X		X	X

Table 3.2: Formal context $\mathbb{K}'_{countries}$.

3.4.1 Using background knowledge

When starting an exploration, all the attribute exploration algorithm knows about the context is the incidence relation of the context. It acquires all the

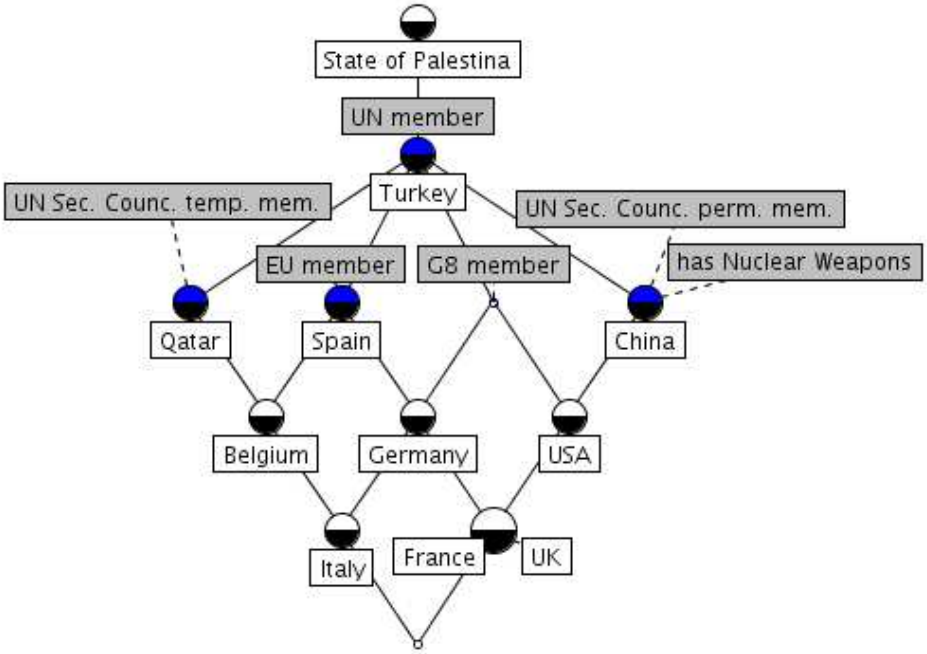

Figure 3.1: Concept lattice of $\mathbb{K}'_{countries}$

necessary knowledge for completing the context from a domain expert. In some applications, before starting the attribute exploration, some of the relations between the attributes is already known. If this background knowledge is implicational, i.e., it it is expressible as Horn clauses, it can easily be incorporated into the attribute exploration algorithm. One can just initialize the set of implications $\mathcal{L}_0$ in Algorithm 2, which is normally empty, with the background implications and start attribute exploration. In this way attribute exploration will work as usual, i.e., it will ask the minimum number of questions to the expert in order to determine the remaining part of the implicational theory. However, in this case the whole set of implications resulting from the exploration, i.e, background implications plus the implications added during the exploration cannot be guaranteed to be of minimum size since the given set of background implications need not necessarily be of minimum size.

In some applications, the restriction to implications can be too strict and a relaxation to non-implicational background knowledge becomes inevitable. For instance, one might want to express that every object satisfies either attribute m_i or attribute m_j. Such background knowledge cannot be expressed by using implications, it requires general propositional formulae. The presence of such background knowledge makes deducing new consequences computationally costly. While reasoning with implications is

polynomial, non-implicational background knowledge makes it NP-complete. In [Gan99, GK05] an extension of attribute exploration that can use non-implicational background knowledge has been introduced. The approach is based on a set of propositional formulae called *cumulated clauses*, which are conjunctions of clauses that have the same negated part. Cumulated clauses generalize the usual clauses of propositional logic. In [GK05] it is shown that the suggested extension of the attribute exploration generates an irredundant basis. However, unlike in the case of implications, the basis generated by this algorithm is not necessarily of minimum cardinality. There it is also shown that the complexity of implicational inference, modulo background knowledge given in the form of cumulated clauses, depends linearly on the implicational part.

Chapter 4

Supporting Bottom-up Construction

In the present chapter we describe how tools from Formal Concept Analysis can be used to support the bottom-up construction of Description Logic knowledge bases. We address the problem of supporting bottom-up construction in a general setting where the user is allowed to re-use concepts from an existing knowledge base that is written in an expressive Description Logic. In Section 4.1 we introduce this extended bottom-up approach. In Section 4.2 we introduce a new non-standard inference, namely least common subsumer w.r.t. a background terminology, which is required in the extended bottom-up approach. In Section 4.3 we follow a more practical approach. We describe a method for computing "good" common subsumers w.r.t. background TBoxes which may not be the least common subsumers, but which are better than the common subsumers computed by ignoring the TBox. As a tool, this method uses the attribute exploration method of Formal Concept Analysis. In Section 4.4 we describe the use of attribute exploration for this setting and report on the first experimental results. In Sections 4.6 and 4.7 we recall two other uses of attribute exploration in the literature for solving problems occurring in a similar setting. We conclude with Section 4.8, where we show that, from a lattice-theoretic point of view, the approaches in Sections 4.4, 4.6, and 4.7 are actually instances of a more general approach.

4.1 The extended bottom-up approach

Traditionally, DL knowledge bases are built in a top-down manner, in the sense that first the relevant notions of the domain are formalized by concept descriptions, and then these concept descriptions are used to specify properties of the individuals occurring in the domain. However, this top-down approach is not always adequate. On the one hand, it might not always be intuitive which notions of the domain are the relevant ones for a particular

application. On the other hand, even if this is intuitive, it might not always be easy to come up with a clear formal description of these notions, especially for a domain expert who is not an expert in knowledge engineering. In order to overcome this, in [BK98, BKM99] a "bottom-up approach" was introduced for constructing DL knowledge bases. In this approach, instead of directly defining a new concept, the domain expert introduces several typical examples as objects, which are then automatically generalized into a concept description by the system. This description is then offered to the domain expert as a possible candidate for a definition of the concept. A successful application of this method was presented in the field of chemical process engineering in [BT01]. The task of computing such a concept description can be split into two subtasks:

- computing the most specific concepts of the given objects,
- and then computing the least common subsumer of these concepts.

The *most specific concept* (msc) of an object o is the most specific concept description C expressible in the given DL language that has o as an instance. The *least common subsumer* (lcs) of concept descriptions $C_1, \ldots, C_n$ is the most specific concept description C expressible in the given DL language that subsumes $C_1, \ldots, C_n$. The problem of computing the lcs and (to a more limited extent) the msc has already been investigated in the literature [CH94, FP96, BK98, BKM99, KM01a, KM01b, KB01, Baa03a, Baa03b, Baa03c, Bra06].

The methods for computing the least common subsumer are restricted to rather inexpressive descriptions logics not allowing for disjunction (and thus not allowing for full negation). In fact, for languages with disjunction, the lcs of a collection of concepts is just their disjunction, and nothing new can be learned from building it. In contrast, for languages without disjunction, the lcs extracts the "commonalities" of the given collection of concepts. Modern DL systems like FaCT [Hor98], RACER [HM01b] and Pellet [SP04] are based on very expressive DLs, and there exist large knowledge bases that use this expressive power and can be processed by these systems [RH97, SH00, HM01a]. In order to allow the user to re-use concepts defined in such existing knowledge bases and still support the user during the definition of new concepts with the bottom-up approach sketched above, we propose the following *extended bottom-up approach*: assume that there is a fixed *background terminology* defined in an expressive DL; e.g., a large ontology written by experts which the user has bought from some ontology provider. The user then wants to extend this terminology in order to adapt it to the needs of a particular application domain. However, since the user is not a DL expert, he employs a less expressive DL and needs support through the bottom-up approach when building this user-specific extension of the background terminology. There are several reasons for the user to employ

a restricted DL in this setting: first, such a restricted DL may be easier to comprehend and use for a non-expert; second, it may allow for a more intuitive graphical or frame-like user interface; third, to use the bottom-up approach, the lcs must exist and make sense, and it must be possible to compute it with reasonable effort.

To make this more precise, consider a background terminology (TBox) $\mathcal{T}$ defined in an expressive DL $\mathcal{L}_2$. When defining new concepts, the user employs only a sublanguage $\mathcal{L}_1$ of $\mathcal{L}_2$ for which computing the lcs makes sense. However, in addition to primitive concepts and roles, the concept descriptions written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$. Let us call such concept descriptions $\mathcal{L}_1(\mathcal{T})$-concept descriptions. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, we want to compute their lcs in $\mathcal{L}_1(\mathcal{T})$, i.e., the least $\mathcal{L}_1(\mathcal{T})$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. In Section 4.2 we consider the case where $\mathcal{L}_1$ is the DL $\mathcal{ALE}$ and $\mathcal{L}_2$ is the DL $\mathcal{ALC}$ and report the following two results from [BST04a, BST07]:

- If $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox, then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept descriptions always exists.
- If $\mathcal{T}$ is a general $\mathcal{ALC}$-TBox allowing for general concept inclusion axioms (GCIs), then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept descriptions need not exist.

The first result on the existence and computability of the lcs w.r.t. an acyclic background terminology is theoretical in the sense that it does not yield a practical algorithm. Due to this, in Section 4.3 we develop a more practical approach. Assume that $\mathcal{L}_1$ is a DL for which least common subsumers (without background TBox) always exist. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can compute a common subsumer w.r.t. $\mathcal{T}$ by just ignoring $\mathcal{T}$, i.e., by treating the defined names in $C_1, \ldots, C_n$ as primitive and computing the lcs of $C_1, \ldots, C_n$ in $\mathcal{L}_1$. However, the common subsumer obtained this way will usually be too general. We sketch a practical method for computing "good" common subsumers w.r.t. background TBoxes, which may not be the *least* common subsumers, but which are more specific than the common subsumers computed by ignoring the TBox. As a tool, this method uses attribute exploration [Gan84] (possibly with background knowledge [Gan99, GK99, GK05]). The application of attribute exploration for this purpose is described in Section 4.4.

4.2 Computing LCS w.r.t. background terminology

In Section 2.4 we have introduced the non-standard inference problem of computing the lcs. However, for the extended bottom-up approach described

above, we need to compute the lcs of a set of concept descriptions with respect to a background TBox. Let us now define this new non-standard inference, which is called *lcs w.r.t. a background terminology*. It is a generalization of the usual lcs to (a)cyclic or general background TBoxes. In the following let $\mathcal{L}_1, \mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sublanguage of $\mathcal{L}_2$, i.e., $\mathcal{L}_1$ allows for less constructors. For a given $\mathcal{L}_2$-TBox $\mathcal{T}$, we call $\mathcal{L}_1(\mathcal{T})$-*concept descriptions* those $\mathcal{L}_1$-concept descriptions that may contain, amongst others, concepts defined in $\mathcal{T}$.

Definition 4.2.1. Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, the *least common subsumer* (lcs) of $C_1, \ldots, C_n$ in $\mathcal{L}_1(\mathcal{T})$ *w.r.t.* $\mathcal{T}$ is the most specific $\mathcal{L}_1(\mathcal{T})$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$, i.e., it is an $\mathcal{L}_1(\mathcal{T})$-concept description D such that

1. $C_i \sqsubseteq_{\mathcal{T}} D$ for $i = 1, \ldots, n$ *(D is a common subsumer);*
2. if E is an $\mathcal{L}_1(\mathcal{T})$-concept description satisfying $C_i \sqsubseteq_{\mathcal{T}} E$ for $i = 1, \ldots, n$, then $D \sqsubseteq_{\mathcal{T}} E$ *(D is least).*

◇

Depending on the DLs $\mathcal{L}_1$ and $\mathcal{L}_2$, least common subsumers of $\mathcal{L}_1(\mathcal{T})$-concept descriptions w.r.t. an $\mathcal{L}_2$-TBox $\mathcal{T}$ may exist or not. Note that this lcs may use only concept constructors from $\mathcal{L}_1$, but may also contain concept names defined in the $\mathcal{L}_2$-TBox. This is the main distinguishing feature of this new notion of a least common subsumer w.r.t. a background terminology. Let us illustrate this by a trivial example.

Example 4.2.2. Assume that $\mathcal{L}_1$ is the DL $\mathcal{ALE}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. Consider the $\mathcal{ALC}$-TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$, and assume that we want to compute the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions P and Q. Obviously, A is the lcs of P and Q w.r.t. $\mathcal{T}$. If we were not allowed to use the name A defined in $\mathcal{T}$, then the only common subsumer of P and Q in $\mathcal{ALE}$ would be the top-concept $\top$. ◇

At first sight, one might think that, in the case of an acyclic background TBox, the problem of computing the lcs in $\mathcal{ALE}(\mathcal{T})$ w.r.t. an $\mathcal{ALC}$-TBox $\mathcal{T}$ can be reduced to the problem of computing the lcs in $\mathcal{ALE}$ by expanding the TBox and using results on the approximation of $\mathcal{ALC}$ by $\mathcal{ALE}$ [BKT02]. To make this more precise, we must introduce the non-standard inference of approximating concept descriptions of one DL by descriptions of another DL. Let $\mathcal{L}_1, \mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sublanguage of $\mathcal{L}_2$.

Definition 4.2.3. Given an $\mathcal{L}_2$-concept description C, the $\mathcal{L}_1$-concept description D *approximates* C *from above* iff D is the least $\mathcal{L}_1$-concept description satisfying $C \sqsubseteq D$. ◇

In [BKT02] it is shown that the approximation from above of an $\mathcal{ALC}$-concept description by an $\mathcal{ALE}$-concept description always exists, and can be computed in double-exponential time.

Thus, given an acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$ and a collection of $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can first *expand* $C_1, \ldots, C_n$ into concept descriptions $C_1', \ldots, C_n'$ by iteratively replacing defined concepts by their definitions until they contain no defined concepts. These descriptions are $\mathcal{ALC}$-concept descriptions since they may contain constructors of $\mathcal{ALC}$ that are not allowed in $\mathcal{ALE}$. One can then build the $\mathcal{ALC}$-concept description $C := C_1' \sqcup \ldots \sqcup C_n'$, and finally *approximate* C from above by an $\mathcal{ALE}$-concept description D. By construction, D is a common subsumer of $C_1, \ldots, C_n$.

However, D does not contain concept names defined in $\mathcal{T}$, and thus it is not necessarily the *least* $\mathcal{ALE}(\mathcal{T})$-concept description subsuming $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. Indeed, this is the case in Example 4.2.2 above, where the approach based on approximation that we have just sketched yields $\top$ rather than the lcs A. One might now think that this can be overcome by applying known results on rewriting concept descriptions w.r.t. a terminology [BKM00]. However, in Example 4.2.2, the concept description $\top$ cannot be rewritten using the TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$.

How can we compute the lcs w.r.t. a background terminology? Does it exist at all? The following result (Theorem 12 in [BST07]) shows that the lcs of $\mathcal{ALE}(\mathcal{T})$-concept descriptions exist, and can effectively be computed. The proof of this result is beyond the scope of this work. For the proof see [BST07] or the forthcoming Ph.D. thesis by Turhan [Tur07].

Theorem 4.2.4. *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox. The lcs of $\mathcal{ALE}(\mathcal{T})$-concept descriptions w.r.t. $\mathcal{T}$ always exists and can effectively be computed.*

The problem of computing the lcs w.r.t. a background terminology was first considered in [BST04b], where $\mathcal{L}_1$ was chosen as the DL $\mathcal{EL}$, and $\mathcal{L}_2$ was chosen to be $\mathcal{ALC}$. There it was shown that:

- If $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox, then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{EL}(\mathcal{T})$-concept descriptions always exists.
- If $\mathcal{T}$ is a general $\mathcal{ALC}$-TBox allowing for general concept inclusion axioms (GCIs), then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{EL}(\mathcal{T})$-concept descriptions need not exist.

4.3 Good common subsumers

Unfortunately, the proof of Theorem 4.2.4 above does not lead to an efficient way of computing the lcs w.r.t. a background terminology. The brute-force algorithm for computing the lcs in $\mathcal{ALE}(\mathcal{T})$ w.r.t. an acyclic background $\mathcal{ALC}$-TBox described in [BST07] in the proof of this theorem is not useful

in practice since the number of concept descriptions that must be considered is very large (super-exponential in the role depth). In addition, w.r.t. cyclic or general TBoxes the lcs need not exist.

In the bottom-up construction of DL knowledge bases, it is not really necessary to take the *least* common subsumer. Using the least one may even result in over-fitting, which is an unwanted effect. Instead, a common subsumer that is not too general can also be used. Now we introduce an approach for computing such *"good"* common subsumers w.r.t. a background TBox. In order to explain this approach, we must first recall how the lcs of $\mathcal{ALE}$-concept descriptions (without background TBox) can be computed.

4.3.1 The LCS of $\mathcal{ALE}$-concept descriptions

Since the lcs of n concept descriptions can be obtained by iterating the application of the binary lcs, we describe how to compute the least common subsumer $\mathsf{lcs}_{\mathcal{ALE}}(C, D)$ of two $\mathcal{ALE}$-concept descriptions C, D (see [BKM99] for more details and a proof of correctness).

First, the input descriptions C, D are normalized by applying the following equivalence-preserving rules modulo associativity and commutativity of conjunction:

$$\begin{array}{rcl} \forall r.E \sqcap \forall r.F & \longrightarrow & \forall r.(E \sqcap F), \\ \forall r.\top & \longrightarrow & \top, \\ \exists r.\bot & \longrightarrow & \bot, \\ A \sqcap \neg A & \longrightarrow & \bot \end{array} \qquad \begin{array}{rcl} \forall r.E \sqcap \exists r.F & \longrightarrow & \forall r.E \sqcap \exists r.(E \sqcap F), \\ E \sqcap \top & \longrightarrow & E, \\ E \sqcap \bot & \longrightarrow & \bot, \end{array}$$

where E and F are concept descriptions, and A is a concept name.

Note that, due to the second rule in the first line, this normalization may lead to an exponential blow-up of the concept descriptions. In the following, we assume that the input descriptions C, D are normalized.

In order to describe the lcs algorithm, we need to introduce some notation. Let C be a normalized $\mathcal{ALE}$-concept description. Then $\mathsf{names}(C)$ ($\overline{\mathsf{names}}(C)$) denotes the set of (negated) concept names occurring in the top-level conjunction of C, $\mathsf{roles}^{\exists}(C)$ ($\mathsf{roles}^{\forall}(C)$) the set of role names occurring in an existential (value) restriction on the top-level of C, and $\mathsf{restrict}^{\exists}_r(C)$ ($\mathsf{restrict}^{\forall}_r(C)$) denotes the set of all concept descriptions occurring in an existential (value) restriction on the role r in the top-level conjunction of C.

Now, let C, D be normalized $\mathcal{ALE}$-concept descriptions. If C (D) is equivalent to $\bot$, then $\mathsf{lcs}_{\mathcal{ALE}}(C, D) = D$ ($\mathsf{lcs}_{\mathcal{ALE}}(C, D) = C$). Otherwise, we have

$$\begin{aligned} \mathsf{lcs}_{\mathcal{ALE}}(C, D) \;=\; & \textstyle\bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \;\sqcap\; \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)} \neg B \;\sqcap \\ & \textstyle\bigsqcap_{r \in \mathsf{roles}^{\exists}(C) \cap \mathsf{roles}^{\exists}(D)} \bigsqcap_{E \in \mathsf{restrict}^{\exists}_r(C), F \in \mathsf{restrict}^{\exists}_r(D)} \exists r.\mathsf{lcs}_{\mathcal{ALE}}(E, F) \;\sqcap \\ & \textstyle\bigsqcap_{r \in \mathsf{roles}^{\forall}(C) \cap \mathsf{roles}^{\forall}(D)} \bigsqcap_{E \in \mathsf{restrict}^{\forall}_r(C), F \in \mathsf{restrict}^{\forall}_r(D)} \forall r.\mathsf{lcs}_{\mathcal{ALE}}(E, F) \;. \end{aligned}$$

Here, the empty conjunction stands for the top-concept $\top$. The recursive calls of $\mathsf{lcs}_{\mathcal{ALE}}$ are well-founded since the role depth decreases with each call.

4.3.2 GCS in $\mathcal{ALE}$ w.r.t. a background terminology

Let $\mathcal{T}$ be a background TBox in some DL $\mathcal{L}_2$ extending $\mathcal{ALE}$ such that subsumption in $\mathcal{L}_2$ w.r.t. this kind of TBoxes is decidable.[1] Let C, D be normalized $\mathcal{ALE}(\mathcal{T})$-concept descriptions. If we ignore the TBox, then we can simply apply the above algorithm for $\mathcal{ALE}$-concept descriptions without background terminology to compute a common subsumer. However, in this context, taking

$$\bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \;\sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)} \neg B$$

is not the best we can do. In fact, some of these concept names may be constrained by the TBox, and thus there may be relationships between them that we ignore by simply using their conjunction. Instead, we propose to take the smallest (w.r.t. subsumption w.r.t. $\mathcal{T}$) conjunction of concept names and negated concept names that subsumes (w.r.t. $\mathcal{T}$) both

$$\bigsqcap_{A \in \mathsf{names}(C)} A \;\sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C)} \neg B \quad \text{and} \quad \bigsqcap_{A' \in \mathsf{names}(D)} A' \;\sqcap \bigsqcap_{\neg B' \in \overline{\mathsf{names}}(D)} \neg B'.$$

We modify the above lcs algorithm in this way, not only on the top-level of the input concepts, but also in the recursive steps. It is easy to show that the $\mathcal{ALE}(\mathcal{T})$-concept description computed by this modified algorithm still is a common subsumer of A, B w.r.t. $\mathcal{T}$.

Proposition 4.3.1. *The $\mathcal{ALE}(\mathcal{T})$-concept description E obtained by applying the modified lcs algorithm to $\mathcal{ALE}(\mathcal{T})$-concept descriptions C, D is a common subsumer of C and D w.r.t. $\mathcal{T}$, i.e., $C \sqsubseteq_{\mathcal{T}} E$ and $D \sqsubseteq_{\mathcal{T}} E$.*

In general, this common subsumer will be more specific than the one obtained by ignoring $\mathcal{T}$, though it need not be the least common subsumer. We call the common subsumer computed this way *good common subsumer (gcs)* , and the algorithm that computes it the *gcs algorithm*.

Example 4.3.2. As a simple example, consider the $\mathcal{ALC}$-TBox $\mathcal{T}$:

$$\begin{aligned}
\mathsf{NoSon} &\equiv \forall \mathsf{has\text{-}child}.\mathsf{Female},\\
\mathsf{NoDaughter} &\equiv \forall \mathsf{has\text{-}child}.\neg\mathsf{Female},\\
\mathsf{SonRichDoctor} &\equiv \forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcup (\mathsf{Doctor} \sqcap \mathsf{Rich})),\\
\mathsf{DaughterHappyDoctor} &\equiv \forall \mathsf{has\text{-}child}.(\neg\mathsf{Female} \sqcup (\mathsf{Doctor} \sqcap \mathsf{Happy})),\\
\mathsf{ChildrenDoctor} &\equiv \forall \mathsf{has\text{-}child}.\mathsf{Doctor},
\end{aligned}$$

[1] Note that the TBox $\mathcal{T}$ used as background terminology may be a general TBox.

and the $\mathcal{ALE}$-concept descriptions

$$\begin{array}{rcl} C & := & \exists \mathsf{has\text{-}child}.(\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}), \\ D & := & \exists \mathsf{has\text{-}child}.(\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}). \end{array}$$

By ignoring the TBox, we obtain the $\mathcal{ALE}(\mathcal{T})$-concept description $\exists \mathsf{has\text{-}child}.\top$ as a common subsumer of C, D. However, if we take into account that both $\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}$ and $\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}$ are subsumed by the concept $\mathsf{ChildrenDoctor}$, then we obtain the more specific common subsumer $\exists \mathsf{has\text{-}child}.\mathsf{ChildrenDoctor}$. The gcs of C, D is even more specific. In fact, the least conjunction of (negated) concept names subsuming both $\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}$ and $\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}$ is

$$\mathsf{ChildrenDoctor} \sqcap \mathsf{DaughterHappyDoctor} \sqcap \mathsf{SonRichDoctor},$$

and thus the gcs of C, D is

$$\exists \mathsf{has\text{-}child}.(\mathsf{ChildrenDoctor} \sqcap \mathsf{DaughterHappyDoctor} \sqcap \mathsf{SonRichDoctor}).$$

The conjunct $\mathsf{ChildrenDoctor}$ is actually redundant since it is implied by the remainder of the conjunction. ◇

In order to implement the gcs algorithm, we must be able to compute the smallest conjunction of (negated) concept names that subsumes two such conjunctions C_1 and C_2 w.r.t. $\mathcal{T}$. In principle, one can compute this smallest conjunction by testing, for every (negated) concept name whether it subsumes both C_1 and C_2 w.r.t. $\mathcal{T}$, and then take the conjunction of those (negated) concept names for which the test was positive. However, this results in a large number of (possibly expensive) calls to the subsumption algorithm for $\mathcal{L}_2$ w.r.t. (general or (a)cyclic) TBoxes. Since, in our application scenario (bottom-up construction of DL knowledge bases w.r.t. a given background terminology), the TBox $\mathcal{T}$ is assumed to be fixed, it makes sense to precompute this information.

In the next section we show how the attribute exploration method [Gan84] of formal concept analysis [GW99] can be used for computing the above-mentioned smallest conjunction, which is required for computing a gcs. Alternative approaches for computing common subsumers, which include an approximation-based method called *acs*, and a subsumption-closure-based method called *scs*, will be discussed in the forthcoming Ph.D. thesis by Turhan [Tur07].

4.4 Using FCA for computing GCSs

In order to obtain a practical gcs algorithm, we must be able to compute in an efficient way the smallest conjunction of (negated) concept names that

subsumes the two conjunctions (w.r.t. $\mathcal{T}$) described in Section 4.3.2 above. As already mentioned above, since in our application scenario (bottom-up construction of DL knowledge bases w.r.t. a given background terminology), the TBox $\mathcal{T}$ is assumed to be fixed, it makes sense to precompute this information. Obviously, a naïve approach that calls the subsumption algorithm for each pair of conjunctions of (negated) concept names is too expensive for TBoxes of realistic sizes. Instead, we propose to use attribute exploration for this purpose.

In order to apply attribute exploration to the task of computing the subsumption hierarchy of conjunctions of (negated) concept names (some of which may be defined concepts in an $\mathcal{L}_2$-TBox $\mathcal{T}$), we define a formal context whose concept lattice is isomorphic to the subsumption hierarchy we are interested in. In general, the subsumption relation induces a partial order, and not a lattice structure on concepts. However, in the case of conjunctions of (negated) concept names, all infima exist, and thus also all suprema, i.e., this hierarchy is a complete lattice.

For the case of conjunctions of concept names (without negated names), this problem was first addressed in [Baa95], where the objects of the context were basically all possible counterexamples to subsumption relationships, i.e., interpretations together with an element of the interpretation domain. The resulting "semantic context" has the disadvantage that an "expert" for this context must be able to deliver such counterexamples, i.e., it is not sufficient to have a simple subsumption algorithm for the DL in question. One needs one that, given a subsumption problem "$C \sqsubseteq D$?", is able to compute a counterexample if the subsumption relationship does not hold, i.e., an interpretation $\mathcal{I}$ and an element d of its domain such that $d \in C^{\mathcal{I}} \setminus D^{\mathcal{I}}$. Since the usual tableau-based subsumption algorithms [BS01] in principle try to generate finite countermodels to subsumption relationships, they can usually be extended such that they yield such an object in case the subsumption relationship does not hold. For instance, in [Baa95], this is explicitly shown for the DL $\mathcal{ALC}$. However, the highly optimized algorithms in systems like FaCT and RACER do not produce such countermodels as output. For this reason, we are interested in a context that has the same attributes and the same concept lattice (up to isomorphism), but for which a standard subsumption algorithm can function as an expert. Such a context was first introduced in [BS04]:

Definition 4.4.1. Let $\mathcal{T}$ be an $\mathcal{L}_2$-TBox. The context $\mathbb{K}_{\mathcal{T}} = (G, M, I)$ is defined as follows:

$$\begin{aligned} G &:= \{C \mid C \text{ is an } \mathcal{L}_2\text{-concept description}\}, \\ M &:= \{A_1, \ldots, A_n\} \text{ is the set of concept names occurring in } \mathcal{T}, \\ I &:= \{(C, A) \mid C \sqsubseteq_{\mathcal{T}} A\}. \end{aligned}$$

◇

Before we can prove that this context has the desired properties, we need to prove the following lemma, which relates subsumption in $\mathcal{T}$ with implications holding in $\mathbb{K}_\mathcal{T}$. In the following, if $B = \{C_{i_1}, \ldots, C_{i_r}\}$ is a set of attributes of $\mathbb{K}_\mathcal{T}$ (i.e., a set of concept names occurring in $\mathcal{T}$), then $\bigsqcap B$ denotes their conjunction, i.e., $C_{i_1} \sqcap \ldots \sqcap C_{i_r}$. For the empty set, we define $\bigsqcap \emptyset := \top$.

Lemma 4.4.2. *Let B_1, B_2 be subsets of M. The implication $B_1 \to B_2$ holds in $\mathbb{K}_\mathcal{T}$ iff $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$.*

Proof. First, we prove the *only if* direction. If the implication $B_1 \to B_2$ holds in $\mathbb{K}_\mathcal{T}$, then this means that the following holds for all objects $C \in G$: if $C \sqsubseteq_\mathcal{T} m$ holds for all $m \in B_1$, then $C \sqsubseteq_\mathcal{T} m$ also holds for all $m \in B_2$. Thus, if we take $\bigsqcap B_1$ as object C, we obviously have $\bigsqcap B_1 \sqsubseteq_\mathcal{T} m$ for all $m \in B_1$, and thus $\bigsqcap B_1 \sqsubseteq_\mathcal{T} m$ for all $m \in B_2$, which shows $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$.

Second, we prove the *if* direction. If $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$, then any object C satisfying $C \sqsubseteq_\mathcal{T} \bigsqcap B_1$ also satisfies $C \sqsubseteq_\mathcal{T} \bigsqcap B_2$ by the transitivity of the subsumption relation. Consequently, if C is subsumed by all concepts in B_1, then it is also subsumed by all concepts in B_2, i.e., if C satisfies all attributes in B_1, it also satisfies all attributes in B_2. This shows that the implication $B_1 \to B_2$ holds in $\mathbb{K}_\mathcal{T}$. □

Using this lemma, we can now prove that the formal context in Definition 4.4.1 above has the desired properties.

Theorem 4.4.3. *The concept lattice of the context $\mathbb{K}_\mathcal{T}$ is isomorphic to the subsumption hierarchy of all conjunctions of subsets of M w.r.t. $\mathcal{T}$.*

Proof. In order to obtain an appropriate isomorphism, we define a mapping π from the formal concepts of the context $\mathbb{K}_\mathcal{T}$ to the set of all (equivalence classes of) conjunctions of subsets of M as follows:

$$\pi(A, B) = [\bigsqcap B]_\equiv.$$

For formal concepts $(A_1, B_1), (A_2, B_2)$ of $\mathbb{K}_\mathcal{T}$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $B_2 \subseteq B_1$. Since B_1 is a concept intent, we have $B_1 = B_1''$, and thus $B_2 \subseteq B_1$ iff $B_2 \subseteq B_1''$ iff the implication $B_1 \to B_2$ holds in $\mathbb{K}_\mathcal{T}$ iff $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$. Overall, we have thus shown that π is an order embedding (and thus injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $[\bigsqcap B_1]_\equiv \sqsubseteq_\mathcal{T}^\equiv [\bigsqcap B_2]_\equiv$.

It remains to show that π is surjective as well. Let B be an arbitrary subset of M. We must show that $[\bigsqcap B]_\equiv$ can be obtained as an image under the mapping π. We know that (B', B'') is a formal concept of $\mathbb{K}_\mathcal{T}$, and thus it is sufficient to show that $\pi(B', B'') = [\bigsqcap B]_\equiv$, i.e., $\bigsqcap B'' \equiv_\mathcal{T} \bigsqcap B$. Obviously, $B \subseteq B''$ implies $\bigsqcap B'' \sqsubseteq_\mathcal{T} \bigsqcap B$. Conversely, the implication $B \to B''$ holds in $\mathbb{K}_\mathcal{T}$, and thus Lemma 4.4.2 yields $\bigsqcap B \sqsubseteq_\mathcal{T} \bigsqcap B''$. □

Now we show that attribute exploration can be used to compute the concept lattice of $\mathbb{K}_{\mathcal{T}}$ since any standard subsumption algorithm for the DL $\mathcal{L}_2$ can function as an expert for the $\mathbb{K}_{\mathcal{T}}$.

Proposition 4.4.4. *Any decision procedure for subsumption w.r.t. TBoxes in $\mathcal{L}_2$ functions as an expert for the context $\mathbb{K}_{\mathcal{T}}$ in Definition 4.4.1.*

Proof. The attribute exploration algorithm asks questions of the form "$B_1 \to B_2$?" By Lemma 4.4.2, we can translate these questions into subsumption questions of the form "$\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$?" Obviously, any decision procedure for subsumption can answer these questions correctly.

Now, assume that $B_1 \to B_2$ does *not* hold in $\mathbb{K}_{\mathcal{T}}$, i.e., $\bigsqcap B_1 \not\sqsubseteq_{\mathcal{T}} \bigsqcap B_2$. We claim that $\bigsqcap B_1$ is a counterexample, i.e., $\bigsqcap B_1 \in B_1'$, but $\bigsqcap B_1 \notin B_2'$. This is an immediate consequence of the facts that $B_i' = \{C \mid C \sqsubseteq_{\mathcal{T}} \bigsqcap B_i\}$ $(i = 1, 2)$, and that $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_1$ and $\bigsqcap B_1 \not\sqsubseteq_{\mathcal{T}} \bigsqcap B_2$.

The expert must provide, for each counterexample, the information which attributes it satisfies and which it does not. This can again be realized by the subsumption algorithm: $\bigsqcap B_1$ satisfies the attribute A_i iff $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} A_i$. □

In order to compute the gcs, we consider not only conjunctions of concept names, but rather conjunctions of concept names *and negated* concept names. The above results can easily be adapted to this case. In fact, one can simply extend the TBox $\mathcal{T}$ by a definition for each negated concept name, and then apply the approach to this extended TBox. To be more precise, if $\{A_1, \ldots, A_n\}$ is the set of concept names occurring in $\mathcal{T}$, then we introduce new concept names $\overline{A}_1, \ldots, \overline{A}_n$, and extend $\mathcal{T}$ to a TBox $\widehat{\mathcal{T}}$ by adding the definitions $\overline{A}_1 \equiv \neg A_1, \ldots, \overline{A}_n \equiv \neg A_n$. Here, for $\widehat{\mathcal{T}}$ to be an $\mathcal{L}_2$-TBox, we must of course assume that $\mathcal{L}_2$ allows for full negation.

Corollary 4.4.5. *The concept lattice of the context $\mathbb{K}_{\widehat{\mathcal{T}}}$ is isomorphic to the subsumption hierarchy of all conjunctions of concept names and negated concept names occurring in $\mathcal{T}$.*

How can this result be used to support computing a gcs? The above corollary together with Proposition 4.4.4 shows that attribute exploration applied to $\mathbb{K}_{\widehat{\mathcal{T}}}$ can be used to compute the Duquenne-Guigues base of $\mathbb{K}_{\widehat{\mathcal{T}}}$. Given this base, we can compute the supremum in the concept lattice of $\mathbb{K}_{\widehat{\mathcal{T}}}$, and thus in the hierarchy of all conjunctions of concept names and negated concept names occurring in $\mathcal{T}$, as follows:

Lemma 4.4.6. *Let $\mathcal{J}$ be the Duquenne-Guigues base of $\mathbb{K}_{\widehat{\mathcal{T}}}$, and let B_1, B_2 be sets of attributes of $\mathbb{K}_{\widehat{\mathcal{T}}}$. Then $\mathcal{J}(B_1) \cap \mathcal{J}(B_2)$ is the intent of the formal concept that is the supremum of the formal concepts (B_1', B_1'') and (B_2', B_2'').*

Proof. We know that the intent of the supremum of the formal concepts (B_1', B_1'') and (B_2', B_2'') is just the intersection $B_1'' \cap B_2''$ of their intents. In addition, since the closure operators $(\cdot)''$ and $\mathcal{J}(\cdot)$ coincide for a formal context, we know that $B_i'' = \mathcal{J}(B_i)$ for $i = 1, 2$. □

As an immediate consequence of this lemma together with Theorem 4.4.3, the supremum in the hierarchy of all conjunctions of concept names and negated concept names occurring in $\mathcal{T}$ can be computed as follows:

Proposition 4.4.7. *Let $\mathcal{J}$ be the Duquenne-Guigues base of $\mathbb{K}_{\widehat{\mathcal{T}}}$, and let B_1, B_2 be sets of (negated) concept names occurring in $\mathcal{T}$. Then*

$$\bigsqcap_{L \in \mathcal{J}(B_1) \cap \mathcal{J}(B_2)} L$$

is the least conjunction of (negated) concept names occurring in $\mathcal{T}$ that lies above both $\bigsqcap_{L \in B_1} L$ and $\bigsqcap_{L \in B_2} L$.

As mentioned in Section 3.3, computing the implicational closure $\mathcal{J}(B)$ for a set of attributes B can be done in linear time in the size of $\mathcal{J}$ and B. This means that the supremum can efficiently be computed.

4.4.1 Using a priori knowledge

When starting the exploration process, all the basic attribute exploration algorithm knows about the context is the set of its attributes. It acquires all the necessary knowledge about the context by asking the expert, which in our setting means: by calling the subsumption algorithm for $\mathcal{L}_2$. Since $\mathcal{L}_2$ is usually an expressive DL, the complexity of the subsumption problem is usually quite high, and thus asking the expert may be expensive. For this reason it makes sense to employ approaches that can avoid some of these expensive calls of the subsumption algorithm.

In our application, we already have some a priori knowledge about relationships between attributes. In fact, we know that the attributes A_i and $\overline{A}_i$ stand for a concept and its negation. In addition, since the background TBox $\mathcal{T}$ is assumed to be an existing terminology, we can usually assume that the subsumption hierarchy between the concept names occurring in $\mathcal{T}$ has already been computed. This provides us with the following *a priori knowledge* about the relationships between attributes:

1. If $A_i \sqsubseteq_{\mathcal{T}} A_j$ holds, then we know on the FCA side that in the context $\mathbb{K}_{\widehat{\mathcal{T}}}$ all objects satisfying the attribute A_i also satisfy the attribute A_j, i.e., the implication $\{A_i\} \to \{A_j\}$ holds in $\mathbb{K}_{\widehat{\mathcal{T}}}$.

2. Since $A_i \sqsubseteq_{\mathcal{T}} A_j$ implies $\neg A_j \sqsubseteq_{\mathcal{T}} \neg A_i$, we also know that all objects satisfying the attribute $\overline{A}_j$ also satisfy the attribute $\overline{A}_i$, i.e., the implication $\{\overline{A}_j\} \to \{\overline{A}_i\}$ holds in $\mathbb{K}_{\widehat{\mathcal{T}}}$.

3. We know that no object can simultaneously satisfy both A_i and $\overline{A}_i$, and thus the implication $\{A_i, \overline{A}_i\} \to \bot^{\mathbb{K}_{\widehat{\mathcal{T}}}}$ holds, where $\bot^{\mathbb{K}_{\widehat{\mathcal{T}}}}$ stands for the set of all attributes of $\mathbb{K}_{\widehat{\mathcal{T}}}$.

4. Every object satisfies either A_i or $\overline{A}_i$. In contrast to the other forms of a priori knowledge mentioned above, this knowledge cannot be encoded in an implication.

The a priori knowledge mentioned in (4) differs from the one mentioned in the other points in that it cannot be represented by Horn clauses. This is the reason why it does not correspond to an implication of the context. In addition, whereas reasoning with respect to Horn clauses (implications) is polynomial, the presence of knowledge of the form mentioned in (4) means that reasoning about the a priori knowledge becomes NP-complete, since it is general propositional reasoning.

Depending on the TBox, there may exist other relationships between attributes that can be deduced, but it should be noted that deducing them makes sense only if this can be done without too much effort: otherwise, the advantage obtained by using the information might be outweighed by the effort of obtaining the a priori knowledge.

An extension of the attribute exploration method called *attribute exploration with background knowledge*, which can use additional information like the one mentioned in (1)–(4) above, was described in [Stu96a, Gan99, GK99, GK05]. This approach can be used to avoid some of the calls to the expert, and thus to speed up the exploration process. In the following, we prefer to use the name "a priori knowledge" instead of background knowledge in order to avoid the confusion with our notion of a background TBox.

If the a priori knowledge is purely implicational (in our case, if we use only the implications mentioned in (1)–(3)), then it is easy to modify the attribute exploration algorithm such that it takes this knowledge into account. In fact, in the initialization phase we simply replace the assignment $\mathcal{J}_0 := \emptyset$ with

$$\begin{aligned} \mathcal{J}_0 \;:=\; & \{\{A_i\} \to \{A_j\} \mid A_i \sqsubseteq_{\mathcal{T}} A_j\} \cup \\ & \{\{\overline{A}_j\} \to \{\overline{A}_i\} \mid A_i \sqsubseteq_{\mathcal{T}} A_j\} \cup \\ & \{\{A_i, \overline{A}_i\} \to \perp^{\mathbb{K}_{\hat{\mathcal{T}}}} \mid i = 1, \ldots, n\}, \end{aligned}$$

where $\perp^{\mathbb{K}_{\hat{\mathcal{T}}}} = \{A_1, \ldots, A_n, \overline{A}_1, \ldots, \overline{A}_n\}$. The effect of this modification is that, when computing the implicational closure $\mathcal{J}_{i+1}(\cdot)$ during attribute exploration, these a priori known implications are also taken into account. The other effect is, of course, that the overall set of implications obtained by the exploration process need not be of minimum cardinality or even free of redundancies.

In principle, non-implicational a priori knowledge (in our case, the one mentioned in (4)) can be utilized in the same way. In this case, the implication hull $\mathcal{J}_{i+1}(\cdot)$ is replaced by the deductive closure, i.e., given a set of computed implications $\mathcal{J}_{i+1}$, the a priori knowledge Γ (which is a finite set of propositional formulae), and a set of attributes B (which we view as propositional variables), we ask which other propositional variables follow

from $\Gamma \cup \Gamma_{\mathcal{J}_{i+1}} \cup \{\bigwedge_{b \in B} b\}$, where $\Gamma_{\mathcal{J}_{i+1}}$ is the set of propositional formulae obtained by translating the implications in $\mathcal{J}_{i+1}$ to the corresponding propositional formulae.

The problem with using the non-implicational knowledge in this way is that computing the deductive closure of propositional formulae is an NP-complete problem, whereas computing the implication closure is polynomial. During the attribute exploration process, it might make sense to use such a more complex closure operator if this saves us calls to the (in general even more complex) subsumption algorithm realizing the expert. However, we also need to use this closure operation when computing the supremum of two conjunctions of (negated) concept names during the gcs computation. Computing the concept lattice is done only once for the given background TBox, whereas the supremum operation is executed several times whenever the user wants the system to compute a gcs. For this reason, the algorithm for computing the supremum operation should be very efficient, and thus we do not want to use full propositional reasoning when computing the supremum.

This does not mean that the non-implicational a priori knowledge cannot be used at all during attribute exploration. For example, one can use it to *"optimize the expert"*. In fact, assume that we use only the implicational knowledge when computing the closure during attribute exploration. If the exploration process generates an implication question "$B_i \to B_i'' \setminus B_i$," then one can first check (by propositional reasoning) whether this implication follows from the implications together with the non-implicational a priori knowledge. If the answer is "yes," then one knows that this is a valid implication. Only if the answer is "no" does one need to call the expert. These propositional pre-tests make sense if the expert is realized by an algorithm that is more complex than the algorithm used for propositional reasoning. Since this approach only optimizes the expert, the set of implications computed by it is identical to the one computed when using only the implicational a priori knowledge. Thus, the supremum can be obtained by computing only the implication closure.

4.4.2 Experimental results

In order to obtain a first impression of the applicability of the gcs algorithm and to identify parts where further optimization is necessary, we have carried out experiments using several small background TBoxes. The reason for using small knowledge bases is that computing the subsumption hierarchy of all conjunctions of (negated) concept names is rather time consuming. In fact, if the TBox contains n concept names (this includes the primitive ones), then the corresponding context has $2n$ attributes, and in the worst case attribute exploration must run through 2^{2n} iterations. Since this is done only once for a given background TBox, long run-times are not prohibitive as

	DICE1	DICE2	DICE3	PA-6	HC	Family
DL	$\mathcal{ALC}$	$\mathcal{ALC}$	$\mathcal{ALC}$	$\mathcal{ALE}$	Boolean	$\mathcal{ALC}$
nr. concept names	10	12	13	12	14	9

Table 4.1: The background TBoxes used in the experiments

long as the computed implication base is rather small, and thus computing the supremum in the gcs algorithm is fast. However, these long run-times would have prevented us from experimenting with different kinds of TBoxes.

The experiments were performed on a computer with one Pentium 4 processor at 2.40 GHz and 2GB of memory, under the GNU/Linux operating system. The implementation was made in the LISP programming language using the version 19a of the CMU Common Lisp interpreter. We used the version 1.7.23 of the Description Logic System RACER [HM01b] as the expert for attribute exploration. For implicational closure calculation, we implemented the linear time implicational closure algorithm *linclosure* presented in Section 4.6 of [Mai83].

The background TBoxes

In order to obtain experimental results that are relevant in practice, we used TBoxes that closely resemble fragments of knowledge bases from applications.

Three such fragments, called DICE1, DICE2, and DICE3 in the following, were obtained from the DICE knowledge base [CAH03], which comes from a medical application and defines concepts from the intensive care domain. The original DICE knowledge base contains more than 2000 concept definitions, is acyclic, and is written in the DL $\mathcal{ALCQ}$, which extends $\mathcal{ALC}$ by so-called qualified number restrictions [HB91]. The TBoxes DICE1, DICE2, and DICE3 were obtained from DICE by selecting a relatively small number of concept definitions and modifying these definitions such that the obtained fragment belongs to $\mathcal{ALC}$ and the number of concept names used in the TBox is small. In addition, two of these TBoxes (DICE2 and DICE3) were modified such that they contain cyclic concept definitions.

A fourth fragment, called PA-6 in the following, was obtained from a process engineering application [SYvWM04]. The original knowledge base describes reactor models and parts of reactors from a polyamid process, consists of about 60 acyclic concept definitions, and is an $\mathcal{ALE}$ knowledge base. Again, we selected a small number of concept definitions from this knowledge base to construct the TBox used in our experiments.

The other two knowledge bases used in our experiments were handcrafted. One is the family TBox from Example 4.3.2, called Family in the

TBox	a.k. type	number of calls			cpu time (secs)			
		expert	imp. hull	pre expert	expert	imp. hull	pre expert	total
DICE1	0	1,309	4,537	-	2.13	1.22	-	23.81
10	1	1,290	3,905	-	2.18	0.72	-	23.78
names	2	1,288	3,905	1,290	1.83	0.70	1.15	21.33
DICE2	0	54,696	132,731	-	91.62	32.44	-	2072.21
12	1	54,678	132,589	-	93.55	22.01	-	2058.08
names	2	54,676	132,589	54,678	92.60	22.54	66.65	2123.30
DICE3	0	91,880	246,616	-	157.66	90.83	-	4862.17
13	1	91,860	246,437	-	154.33	57.78	-	4795.51
names	2	91,856	246,437	91,860	154.96	56.68	183.62	5021.54
PA-6	0	30,484	110,671	-	93.52	55.06	-	943.25
12	1	30,462	95,572	-	52.42	24.69	-	907.22
names	2	30,457	95,572	30,462	50.47	24.84	53.77	927.13
HC	0	4,794	17,816	-	8.19	33.86	-	131.34
14	1	4,776	17,629	-	7.89	19.18	-	112.99
names	2	4,755	17,629	4,776	7.79	19.21	77.35	129.81
Family	0	6,334	16,962	-	9.31	2.22	-	102.89
9	1	6,321	16,905	-	9.74	1.48	-	103.83
names	2	6,319	16,905	6,321	9.22	0.67	2.87	97.81

Table 4.2: Attribute exploration on background TBoxes

following, and the other is a small acyclic TBox, called HC in the following, which uses only Boolean operations and was built such that there are relationships between conjunctions of (negated) concept names that do not follow from the subsumption relationships between the names.

Table 4.1 displays the number of concept names each background TBox contains. Note that the formal context obtained from a background TBox with n concept names has $2n$ attributes since it contains the negated concept names as attributes as well.

Computing the subsumption lattice

We computed the subsumption hierarchy of all conjunctions of (negated) concept names for the six TBoxes mentioned above, using three different variants of Ganter's attribute exploration algorithm:

Type 0: The usual attribute exploration algorithm that does not use any a priori knowledge.

Type 1: The attribute exploration algorithm that uses the implicational a priori knowledge (1)–(3) described in Section 4.4.1.

Type 2: Like Type 1, but now the non-implicational a priori knowledge (4) is used to optimize the expert, as sketched at the end of Section 4.4.1. Propositional consequences are computed using an algorithm by Ganter [GK99, GK05], which is linear in the size of the implicational part of the a priori knowledge, but exponential in the size of the non-implicational part.

Table 4.2 shows the number of calls to the expert, the number of computations of the implication hull during attribute exploration, and the number of calls to the pre-expert realized using non-implicational a priori knowledge. It also shows the time spent in the respective calls where, for the expert, we have only measured the time spent in answering the implication question, but not the time spent to produce the counterexample (since only this part can be addressed by the pre-expert).

The numbers show that using implicational a priori knowledge leads to an improvement of the attribute exploration algorithm since it reduces the number of calls to the expert and the closure computation, and it also reduces the overall run-time. However, these gains are rather moderate. Using non-implicational a priori knowledge in the way we currently do is not advisable: there is only a very moderate reduction of the number of calls to the expert, and the additional calls to the pre-expert often take more time than is gained this way. However, things may change if one uses a highly optimized propositional reasoner to realize the pre-expert, and when calls to the expert become more expensive for larger background TBoxes.

What Table 4.2 also shows is that most of the time spent in attribute exploration is *not* spent answering implication questions or computing the implication closure. The major culprit actually turns out to be the computation of the operation $B_i \mapsto B_i''$. The reason is that the number of objects in the contexts computed during attribute exploration becomes very large. For example, consider the DICE3 TBox. There, we have 91,880 calls to the expert, but only 27 implications (see Table 4.3). This means that in all but 27 cases, the expert produces a counterexample. Thus, the final context consists of 91,853 objects. In addition to the problem caused by the large number of objects when computing the operation $B_i \mapsto B_i''$, the expert also spends quite some time actually producing the counterexample, i.e., checking by subsumption tests which attributes the counterexample satisfies and which it does not. As mentioned above, this time was not measured in the column for the run-time of the expert. Excessive number of calls to the expert and the long runtimes for the TBoxes DICE-2, DICE-3 and PA-6 can be explained by the number of concept names they contain and by the fact that they do not contain much relationships between these concept names. At this point, one can argue that these values are much lower for the TBox HC, although it contains more concept names. As mentioned before, HC is built to contain the specific kind of relationships we want to have, so com-

TBox	a.k. type	size of base computed	size of base total	TBox	a.k. type	size of base computed	size of base total
DICE1	0	24	24	PA-6	0	31	31
10 names	1	3	25	12 names	1	7	31
DICE2	0	21	21	HC	0	56	56
12 names	1	3	22	14 names	1	32	62
DICE3	0	27	27	Family	0	16	16
13 names	1	6	28	9 names	1	3	16

Table 4.3: Sizes of the implication bases

puting the required hierarchy for this TBox takes much less time compared to the three TBoxes above.

Table 4.3 shows the sizes of the final set of implications computed by the attribute exploration algorithm. Since the third variant of the attribute exploration algorithm (type 2) only "optimizes" the expert, but does not change the the attribute exploration process, the results for it coincide with the second variant (type 1). Thus, it is not explicitly included in the table. For type 1 we distinguish between the set of all implications and the ones computed during the exploration process. For example, for PA-6 we had 24 implications as a priori knowledge and computed 7 additional implications.

The most interesting result that can be drawn from this table is that the number of implications stays rather small (in particular compared to the huge number of objects in the final context). Consequently, computing implication closure for these sets of implications will be fast (see Section 4.4.2 below).

Computing the supremum

Table 4.4 shows the time spent in supremum calls that were generated during the computation of gcs in our experiments (The experimental results concerning the computation of gcs are beyond the scope of this work. They are going to be part of the Ph.D. thesis by Turhan [Tur07]). We measured the run-time for three different methods of computing the supremum:

Type 0: The supremum is computed by building the implication closure w.r.t. the implication base computed without using a priori knowledge during attribute exploration.

Type 1: The supremum is computed by building the implication closure w.r.t. the implication base computed using implicational a priori knowledge during attribute exploration.

Type 2: The supremum is computed naïvely using iterated calls of the subsumption algorithm.

TBox	exp. type	supremum calls	cpu time (secs) average	cpu time (secs) total	base size
DICE1	0	67	0.00015	0.01	24
10	1	67	0.00030	0.02	25
names	2	67	0.03045	2.04	–
DICE2	0	121	0.00016	0.02	21
12	1	121	0.00016	0.02	22
names	2	121	0.0433	5.24	–
DICE3	0	101	0.00039	0.04	27
13	1	101	0.00049	0.05	28
names	2	101	0.04030	4.07	–
PA-6	0	237	0.00046	0.11	31
12	1	237	0.00046	0.11	31
names	2	237	0.0481	11.4	–
HC	0	42	0.00166	0.07	56
14	1	42	0.00238	0.1	62
names	2	42	0.06	2.52	–
Family	0	98	0.0001	0.01	16
9	1	98	0.0002	0.02	16
names	2	98	0.03775	3.7	–

Table 4.4: Computing the supremum

Since the implication base computed with a priori knowledge is usually larger than the one without, using the former base takes longer, but in both cases the supremum computation is fast. In contrast, the naïve approach is two orders of magnitude slower. Thus, it really pays off to compute the implication base in advance.

4.5 Faster computation of the subsumption lattice

In Section 4.4.2 we have mentioned the long runtimes encountered in our experiments, even for relatively small background TBoxes. While analyzing Table 4.2, we have noticed that the long runtimes are mainly due to computation of the $(\cdot)''$ operator. The reason for this is that the number of counterexamples generated during attribute exploration becomes very large. For instance, the DICE3 TBox that contains only 13 concept names leads to a formal context with 91,853 objects, which makes the computation of the $(\cdot)''$ operator quite time consuming. In this section we look for a way of overcoming this problem by trying to keep the number of counterexamples small.

For the experiments in Section 4.4.2, we have employed a very basic idea for generating counterexamples: whenever an implication $B_1 \to B_2$ is rejected, start with the concept description $\bigsqcap B_1$. It is definitely a coun-

terexample. Then determine which attributes this counterexample has by iterating over the set $M \setminus B_1$, i.e., for each $A_i \in M \setminus B_1$ if $\bigsqcap B_1 \sqsubseteq_{\widehat{\mathcal{T}}} A_i$, then add A_i to the intent of $\bigsqcap B_1$. However, in this way we do not take into account that the attribute set M also contains the negation of each concept name occurring in the TBox. It might be the case that, for some $1 \leq i \leq n$, neither $\bigsqcap B_1 \sqsubseteq A_i$, nor $\bigsqcap B_1 \sqsubseteq \overline{A}_i$ follows from $\widehat{\mathcal{T}}$. As a result, we generate a counterexample that has neither the attribute A_i nor $\overline{A}_i$. A closer look reveals that such counterexamples are reducible, i.e., they can be removed from the formal context without changing the structure of the concept lattice. One idea could be to reduce the context from time to time during attribute exploration. However, since context reduction is a costly operation, this approach is not feasible. Instead, by employing a more clever idea that takes into account that M also contains an attribute for the negation of each concept name, it is possible to avoid generating reducible counterexamples. The idea is that the counterexamples that have either A_i or $\overline{A}_i$ for each concept name A_i are irreducible, and it is always possible to generate such a counterexample when an implication is rejected. The following theorem formalizes this argument.

Theorem 4.5.1. *Assume that the implication $B_1 \rightarrow B_2$ does not hold in $\mathbb{K}_{\widehat{\mathcal{T}}}$ and $\bigsqcap C$ is a counterexample to this implication such that $\bigsqcap C \not\sqsubseteq_{\widehat{\mathcal{T}}} A_i$ and $\bigsqcap C \not\sqsubseteq_{\widehat{\mathcal{T}}} \overline{A}_i$ for some $1 \leq i \leq n$. Then one of $\bigsqcap C \sqcap A_i$ or $\bigsqcap C \sqcap \overline{A}_i$ is also a counterexample to this implication.*

Proof. According to the argument, $\bigsqcap C \sqcap A_i \not\sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ or $\bigsqcap C \sqcap \neg A_i \not\sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. Assume this were not true, i.e., neither $\bigsqcap C \sqcap A_i$ nor $\bigsqcap C \sqcap \neg A_i$ is a counterexample. Then, both $\bigsqcap C \sqcap A_i \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ and $\bigsqcap C \sqcap \neg A_i \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$ would be true. Then the following would also hold: $(\bigsqcap C \sqcap A_i) \sqcup (\bigsqcap C \sqcap \neg A_i) \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$, which would mean $\bigsqcap C \sqcap (A_i \sqcup \neg A_i) \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$. Equivalently, this would mean $\bigsqcap C \sqsubseteq_{\widehat{\mathcal{T}}} \bigsqcap B_2$, which is a contradiction since we assumed that $B_1 \rightarrow B_2$ does not hold in $\mathbb{K}_{\widehat{\mathcal{T}}}$ and that $\bigsqcap C$ is a counterexample. Thus either $\bigsqcap C \sqcap A_i$, or $\bigsqcap C \sqcap \neg A_i$ is a counterexample to $B_1 \rightarrow B_2$. □

By successive applications of the argument in the proof of Theorem 4.5.1, we can generate irreducible counterexamples. Algorithm 3 describes this idea formally.

Proposition 4.5.2. *Given an implication $B_1 \rightarrow B_2$ that is rejected by the expert, Algorithm 3 terminates and, upon termination, it returns a counterexample to this implication, which has either A_i or $\overline{A}_i$ for each $i \in \{1, \ldots, n\}$.*

Proof. The algorithm iterates over a finite set of indices $\{1, \ldots, n\}$, so it terminates. A valid counterexample to $B_1 \rightarrow B_2$ should contain all the attributes in B_1, and should not contain at least one of the attributes in B_2. The object $\bigsqcap B_1$ is a counterexample to this implication. It has all the

Algorithm 3 Counterexample generation for $\mathbb{K}_{\widehat{\mathcal{I}}}$

Initialization {Assume the implication $B_1 \rightarrow B_2$ does not hold in $\mathbb{K}_{\widehat{\mathcal{I}}}$}
$C := B_1$ {$\bigsqcap B_1$ is a counterexample}
for all $i \in \{1, \ldots, n\}$ **do**
  if $A_i \notin C$ and $\overline{A}_i \notin C$ **then**
    if $\bigsqcap C \sqcap A_i \sqsubseteq_{\widehat{\mathcal{I}}} \bigsqcap B_2$ **then**
      $C = C \cup \{\overline{A}_i\}$
    else
      $C = C \cup \{A_i\}$
    end if
  end if
end for
return $\bigsqcap C$

attributes in B_1, and does not have all the attributes in B_2. So, the algorithm starts generating the counterexample with $C := B_1$. In the iteration, for some $i \in \{1, \ldots, n\}$ assume that $\bigsqcap C \sqcap A_i \sqsubseteq_{\widehat{\mathcal{I}}} \bigsqcap B_2$ does not hold. Then $\bigsqcap C \sqcap A_i$ is a counterexample. So, we add A_i to the intent of $\bigsqcap C$. If $\bigsqcap C \sqcap A_i \sqsubseteq_{\widehat{\mathcal{I}}} \bigsqcap B_2$ holds, then by Theorem 4.5.1, $\bigsqcap C \sqcap \overline{A}_i$ is a counterexample and we add $\overline{A}_i$ to the intent of $\bigsqcap C$. Once we finish iterating over all $i \in \{1, \ldots, n\}$ in this way, the resultant C has either A_i or $\overline{A}_i$ for each i, and $\bigsqcap C$ is a counterexample to $B_1 \rightarrow B_2$. □

4.5.1 Experimental results

In order to see the performance gain obtained by using the new counterexample generation method, we have run a new set of experiments on the background TBoxes mentioned in Section 4.4.2. Table 4.5 below shows the results of these new experiments with the improved counterexample generation method.

When we compare the results of the new experiments given in Table 4.5 to the results of the previous experiments given in Table 4.2, the first thing we notice is the drastic amount of decrease in the number of calls to the expert, thus the CPU time spent by the expert, and the total CPU time for each experiment. For instance, take the results of the experiment on TBox DICE1 with no a priori knowledge (a. k. type 0). The first row of Table 4.5 says that the number of calls to the expert is 96 whereas, according to the first row of Table 4.2 it was 1,309 in the previous experiments. As a result, the CPU time spent by the expert decreases from 2.13 seconds to 0.15 seconds, and the total CPU time spent for this experiment decreases from 23.81 seconds to 1.14 seconds. For this TBox, the improvement in runtime is more than 90%. For the other TBoxes the performance gain obtained is also around 90%. The reason for this drastic amount of improvement

TBox	a. k. type	number of calls			cpu time (secs)			
		expert	imp. hull	pre expert	expert	imp. hull	pre expert	total
DICE1	0	96	3,124	-	0.15	0.37	-	1.14
10	1	75	2,999	-	0.18	0.90	-	1.89
names	2	73	2999	75	0.05	0.93	0.15	2.14
DICE2	0	853	163,375	-	1.52	26.62	-	59.03
12	1	835	163,245	-	1.27	36.30	-	68.17
names	2	833	163,245	835	1.18	37.49	2.56	72.28
DICE3	0	1,115	327,526	-	1.78	81.61	-	147.86
13	1	1,094	327,363	-	1.73	108.40	-	172.77
names	2	1090	327,363	1,094	2.06	107.99	4.16	178.94
PA-6	0	559	77,767	-	0.97	16.04	-	31.87
12	1	535	77,580	-	0.73	26.88	-	41.91
names	2	530	77,580	535	0.75	26.88	1.68	44.28
HC	0	184	21,356	-	0.30	19.25	-	22.43
14	1	160	21,164	-	0.23	35.59	-	38.50
names	2	140	21,164	160	0.22	35.67	3.40	42.68
Family	0	240	16,925	-	0.30	1.49	-	4.84
9	1	227	16,868	-	0.29	1.62	-	4.80
names	2	225	16,868	227	0.40	1.54	0.26	5.28

Table 4.5: Experimental results using the improved counterexample generation method

in the performance becomes clear when we compare the sizes of the formal contexts produced in our previous and new experiments. Table 4.6 gives a comparison of the sizes of the formal contexts obtained in our new and previous experiments. For instance, size of the formal context resulting from the TBox DICE1 decreases from 1285 to 72 in the new experiments. This means that 1213 of the counterexamples we have generated in our previous experiment were reducible, i.e., 94% of the counterexamples we have generated were actually redundant. Similarly, for the other TBoxes, the formal context sizes decrease by around 90%. This is due to the following fact: a counterexample we generate with the new method has either A_i, or $\overline{A}_i$ for each $1 \leq i \leq n$, where n is the size of the attribute set M. The upper bound on the number of such counter examples is 2^n. On the other hand, in our previous method we did not require a counterexample to have either A_i or $\overline{A}_i$ for each i. We only required that for any i, it does not contain A_i and $\overline{A}_i$ together. The upper bound on the number of such counterexamples is 2^{2n-1}. Thus, by exploiting a property specific to the formal context we use in our application, namely the dichotomy of the attributes, we save an exponential number of counterexamples. Moreover, generating this new type of counterexamples does not cost more in the number of subsumption tests, compared to the previous method. Assuming that the implication

context size	DICE1 10 names	DICE2 12 names	DICE3 13 names	PA-6 12 names	HC 14 names	Family 9 names
improved version	72	832	1,088	528	128	224
naive version	1,285	54,675	91,853	30,453	4,738	6,318

Table 4.6: Comparison of context sizes

$B_1 \rightarrow B_2$ does not hold in $\mathbb{K}_{\widehat{\mathcal{T}}}$, the size of B_1 is k, and the size of M is $2n$, we make exactly $n-k$ subsumption tests to generate a counterexample, which is the same number of subsumption tests we make in the previously used counterexample generation algorithm.

4.6 The hierarchy of conjunctions of concepts

In the previous section, we have shown how attribute exploration can be used to efficiently compute the subsumption hierarchy of conjunctions of a set of concept names and the negations of these concept names. For the case of conjunctions of concept names only (without negated names), this approach was first used in [Baa95]. There the main motivation was to compute an extended subsumption hierarchy in which the interaction between defined concepts could easily be seen. This was explained by the following example: Assume the defined concept NoDaughter stands for the people that have no daughters, NoSon stands for the people that have no sons, and NoSmallChild stands for the people that have no small children. Obviously, there is no subsumption relationship between these three concepts. On the other hand, the conjunction NoDaughter$\sqcap$NoSon is subsumed by NoSmallChild, i.e., if an individual a belongs to NoSon and NoDaughter, it also belongs to NoSmallChild. However, this cannot be derived from the information that a belongs to NoSon and NoDaughter by just looking at the subsumption hierarchy. This example demonstrates that some runtime inferences concerning individuals could be sped up by precomputing the subsumption hierarchy not only for defined concepts, but also for all conjunctions of defined concepts.

In [Baa95], the attributes of the formal context defined with the above motivation were again concept names (this time without negated names), but the objects were basically all possible counterexamples to subsumption relationships, i.e., interpretations together with an element of the interpretation domain. The disadvantage of this "semantic context" is that a usual subsumption algorithm can not be used as expert for this context within attribute exploration. For this context, one needs a subsumption algorithm that, given a subsumption problem "$C \sqsubseteq D$?", is able to compute an interpretation $\mathcal{I}$ and an element d of its domain such that $d \in C^{\mathcal{I}} \setminus D^{\mathcal{I}}$, whenever the subsumption relationship does not hold. Since the tableau-based sub-

sumption algorithms [BS01] in principle try to generate finite representations of countermodels to subsumption relationships, they can usually be extended such that they yield such a counterexample in case the subsumption relationship does not hold. For instance in [Baa95], this is explicitly shown for the DL $\mathcal{ALC}$. Below we first give the definition of this context called semantic context, and state some results about this context first shown in [Baa95]. Here we omit the proofs of these results. For the proofs see [Baa95] or [BS04] where the approach was reconsidered and a more abstract point of view was presented.

4.6.1 The semantic context

Definition 4.6.1. Given a finite set of concept descriptions $\mathcal{C} = \{C_1, \ldots, C_n\}$, the corresponding context $\mathbb{K}_\mathcal{T} = (G, M, I)$ is defined as follows:

$$\begin{aligned} G &:= \{(\mathcal{I}, d) \mid \mathcal{I} \text{ is a model of } \mathcal{T} \text{ and } d \in \Delta^\mathcal{I}\}, \\ M &:= \mathcal{C}, \\ I &:= \{((\mathcal{I}, d), C_i) \mid d \in C_i^\mathcal{I}\}. \end{aligned}$$

$\diamond$

For a nonempty subset $B = \{C_{i_1}, \ldots, C_{i_r}\}$ of M, we denote the conjunction $C_{i_1} \sqcap \ldots \sqcap C_{i_r}$ by $\bigsqcap B$. For the empty set, we define $\bigsqcap \emptyset := \top$.

The following lemma establishes a correspondence between the implications that hold in the semantic context $\mathbb{K}_\mathcal{T}$ and the subsumption relationships that hold in $\mathcal{T}$.

Lemma 4.6.2. *Let B_1, B_2 be subsets of M. The implication $B_1 \to B_2$ holds in $\mathbb{K}_\mathcal{T}$ iff $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$.*

Thus, the Duquenne-Guigues base $\mathcal{L}$ of $\mathbb{K}_\mathcal{T}$ also yields a representation of all subsumption relationships of the form $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$ for subsets B_1, B_2 of M. As mentioned in Section 3.3, any question "$\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$?" can then be answered in time linear in the size of $\mathcal{L} \cup \{B_1 \to B_2\}$.

Theorem 4.6.3 below shows that $\mathbb{K}_\mathcal{T}$ has the desired property.

Theorem 4.6.3. *The concept lattice of the context $\mathbb{K}_\mathcal{T}$ is isomorphic to the subsumption hierarchy of all conjunctions of subsets of M w.r.t. $\mathcal{T}$.*

If we want to use attribute exploration (Algorithm 2 in Section 3.4) to compute the concept lattice and the Duquenne-Guigues base of $\mathbb{K}_\mathcal{T}$, we need an expert for the context $\mathbb{K}_\mathcal{T}$. This expert must be able to answer the questions asked by the attribute exploration algorithm, i.e., given an implication $B_1 \to B_2$, it must be able to decide whether this implication holds in $\mathbb{K}_\mathcal{T}$. If the implication does not hold, it must be able to compute a counterexample, i.e., an object $o \in B_1' \setminus B_2'$.

By Lemma 4.6.2, $B_1 \rightarrow B_2$ holds in $\mathbb{K}_{\mathcal{T}}$ iff $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$. Thus, validity of implications in $\mathbb{K}_{\mathcal{T}}$ can be decided using a standard subsumption algorithm. A counterexample to the implication $B_1 \rightarrow B_2$ is a pair $(d, \mathcal{I}) \in G$ such that $d \in \bigsqcap B_1^{\mathcal{I}} \setminus \bigsqcap B_2^{\mathcal{I}}$. As already mentioned, since the usual tableau-based subsumption algorithms [BS01] in principle try to generate finite countermodels to subsumption relationships, they can usually be extended such that they yield such an object in case the subsumption relationship does not hold. In [Baa95], this was explicitly shown for the DL $\mathcal{ALC}$.

Proposition 4.6.4. *The tableau-based subsumption algorithm for $\mathcal{ALC}$ Tboxes $\mathcal{T}$ can be extended such that it functions as an expert for the context $\mathbb{K}_{\mathcal{T}}$ without increasing its worst-case complexity of* PSPACE.

However, the highly optimized algorithms in systems like FaCT, RACER and Pellet do not produce such countermodels as output. For this reason, in [BS04] we have introduced a formal context called "syntactic context" that also has the desired property in Theorem 4.6.3, but for which a standard subsumption algorithm can function as expert.

4.6.2 The syntactic context

Definition 4.6.5. Given a finite set of concept descriptions $\mathcal{C} = \{C_1, \ldots, C_n\}$, the corresponding context $\mathbb{K}_{\mathcal{T}} = (G, M, I)$ is defined as follows:

$$\begin{aligned} G &:= \{D \mid D \text{ is a concept description of the DL under consideration}\}; \\ M &:= \mathcal{C}, \\ I &:= \{(D, C_i) \mid D \sqsubseteq_{\mathcal{T}} C_i\}. \end{aligned}$$

◇

The context $\mathbb{K}_{\mathcal{T}}$ satisfies the analogs of Lemma 4.6.2 and Theorem 4.6.3.

Lemma 4.6.6. *Let B_1, B_2 be subsets of M. The implication $B_1 \rightarrow B_2$ holds in $\mathbb{K}_{\mathcal{T}}$ iff $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$.*

Proof. First, we prove the *only if* direction. If the implication $B_1 \rightarrow B_2$ holds in $\mathbb{K}_{\mathcal{T}}$, then this means that the following holds for all objects $D \in G$: if $D \sqsubseteq_{\mathcal{T}} C$ holds for all $C \in B_1$, then $D \sqsubseteq_{\mathcal{T}} C$ also holds for all $C \in B_2$. Thus, if we take $\bigsqcap B_1$ as object D, we obviously have $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} C$ for all $C \in B_1$, and thus $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} C$ for all $C \in B_2$, which shows that $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$.

Second, we prove the *if* direction. If $\bigsqcap B_1 \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$, then any object D satisfying $D \sqsubseteq_{\mathcal{T}} \bigsqcap B_1$ also satisfies $D \sqsubseteq_{\mathcal{T}} \bigsqcap B_2$ by the transitivity of the subsumption relation. Consequently, if D is a subconcept of all concepts in B_1, then it is also a subconcept of all concepts in B_2, i.e., if D satisfies all attributes in B_1, it also satisfies all attributes in B_2. This shows that the implication $B_1 \rightarrow B_2$ holds in $\mathbb{K}_{\mathcal{T}}$. □

Now we show that $\mathbb{K}_\mathcal{T}$ also has the property that its concept lattice is isomorphic to the subsumption hierarchy of all conjunctions of subsets of M.

Theorem 4.6.7. *The concept lattice of the context $\mathbb{K}_\mathcal{T}$ is isomorphic to the subsumption hierarchy of all conjunctions of subsets of M w.r.t. $\mathcal{T}$.*

Proof. We define a mapping π from the formal concepts of the context $\mathbb{K}_\mathcal{T}$ to the set of all (equivalence classes of) conjunctions of subsets of M as follows:

$$\pi(A, B) = [\bigsqcap B]_\equiv.$$

For formal concepts $(A_1, B_1), (A_2, B_2)$ of $\mathbb{K}_\mathcal{T}$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $B_2 \subseteq B_1$. Since B_1 is an intent, we have $B_1 = B_1''$, and thus $B_2 \subseteq B_1$ iff $B_2 \subseteq B_1''$ iff the implication $B_1 \to B_2$ holds in $\mathbb{K}_\mathcal{T}$ iff $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$. Overall, we have thus shown that π is an order embedding (and thus injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $[\bigsqcap B_1]_\equiv \sqsubseteq_\equiv [\bigsqcap B_2]_\equiv$.

It remains to be shown that π is surjective as well. Let B be an arbitrary subset of M. We must show that $[\bigsqcap B]_\equiv$ can be obtained as an image under the mapping π. We know that (B', B'') is a formal concept of $\mathbb{K}_\mathcal{T}$, and thus it is sufficient to show that $\pi(B', B'') = [\bigsqcap B]_\equiv$, i.e., $\bigsqcap B'' \equiv_\mathcal{T} \bigsqcap B$. Obviously, $B \subseteq B''$ implies $\bigsqcap B'' \sqsubseteq_\mathcal{T} \bigsqcap B$. Conversely, the implication $B \to B''$ holds in $\mathbb{K}_\mathcal{T}$, and thus Lemma 4.6.6 yields $\bigsqcap B \sqsubseteq_\mathcal{T} \bigsqcap B''$. □

Again, attribute exploration can be used to compute the concept lattice since any standard subsumption algorithm for the DL under consideration can be used as an expert for $\mathbb{K}_\mathcal{T}$.

Proposition 4.6.8. *Any decision procedure for subsumption functions as an expert for the context $\mathbb{K}_\mathcal{T}$.*

Proof. The attribute exploration algorithm asks questions of the form "$B_1 \to B_2$?" By Lemma 4.6.6, we can translate these questions into subsumption questions of the form "$\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_2$?" Obviously, any decision procedure for subsumption can answer these questions correctly.

Now, assume that $B_1 \to B_2$ does *not* hold in $\mathbb{K}_\mathcal{T}$, i.e., $\bigsqcap B_1 \not\sqsubseteq_\mathcal{T} \bigsqcap B_2$. We claim that $\bigsqcap B_1$ is a counterexample, i.e., $\bigsqcap B_1 \in B_1'$, but $\bigsqcap B_1 \notin B_2'$. This is an immediate consequence of the facts that $B_i' = \{D \mid D \sqsubseteq_\mathcal{T} \bigsqcap B_i\}$ $(i = 1, 2)$ and that $\bigsqcap B_1 \sqsubseteq_\mathcal{T} \bigsqcap B_1$ and $\bigsqcap B_1 \not\sqsubseteq_\mathcal{T} \bigsqcap B_2$. □

4.7 The hierarchy of least common subsumers

In the previous sections, we have shown how attribute exploration can be used to support bottom-up construction of knowledge bases by efficiently computing the subsumption hierarchy of the conjunctions of a set of concept names. Apart from this, another usage of attribute exploration was

presented in [BM00] again to support bottom-up construction of knowledge bases, but this time by computing the hierarchy of least common subsumers of the subsets of a given set of concept descriptions. There, the problem attacked was the choice of examples, which is a crucial problem for the quality of the result obtained by bottom-up construction. In bottom-up construction, if the examples are too similar, the resulting concept might be too specific. Conversely, if the examples are too different, the resulting concept is likely to be too general. Thus, it would be good to have a tool that supports the process of choosing an appropriate set of objects as examples. Assume that $C_1, \ldots, C_n$ are the most specific concepts of a given collection of objects $o_1, \ldots, o_n$, and that we intend to use subsets of this collection for constructing new concepts. In order to avoid obtaining concepts that are too general or too specific, it would be good to know the position of the corresponding lcs in the subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. Since there are exponentially many subsets to be considered, and (depending on the DL language) both, computing the lcs and testing for subsumption, can be expensive operations, we want to obtain complete information on how this hierarchy looks like without computing the least common subsumers of all subsets of $\{C_1, \ldots, C_n\}$, and without explicitly making all the subsumption tests between these least common subsumers. In order to solve this problem, in [BM00], a formal context was defined whose concept lattice is isomorphic to the *inverse* subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. It was shown that attribute exploration can again be used to compute this lattice, and the expert required by the algorithm can be realized by the subsumption algorithm and the algorithm for computing the lcs. Below, we are first going to give the definition of such a formal context, and state results from [BM00] without giving their proofs. For the proofs, the reader is referred to [BM00].

In the following we assume that, in the DL $\mathcal{L}$ under consideration, the lcs always exists and can be effectively computed. Given a finite set $\mathcal{C} := \{C_1, \ldots, C_n\}$ of concept descriptions, we are interested in the subsumption hierarchy between all least common subsumers of subsets of $\mathcal{C}$. For sets $B \subseteq \mathcal{C}$ of cardinality larger than two, we have already defined the notion $\mathsf{lcs}(B)$. We extend this notion to the empty set and singletons in the obvious way: $\mathsf{lcs}(\emptyset) := \bot$ and $\mathsf{lcs}(\{C_i\}) := C_i$.

Our goal is to compute the subsumption hierarchy between all concept descriptions $\mathsf{lcs}(B)$ for subsets B of $\mathcal{C}$ without explicitly computing all these least common subsumers. This is again achieved by defining a formal context (with attribute set $\mathcal{C}$) such that the concept lattice of this context is isomorphic to the subsumption hierarchy we are interested in. The following context is similar to the syntactic context defined in the previous section. The main difference is the definition of the incidence relation, where subsumption is used in the opposite direction.

Definition 4.7.1. Given a DL language $\mathcal{L}$ and a finite set $\mathcal{C} := \{C_1, \ldots, C_n\}$ of $\mathcal{L}$-concept descriptions, the corresponding formal context $\mathbb{K}_{\mathcal{L}}(\mathcal{C}) = (G, M, I)$ is defined as follows:

$$\begin{aligned} G &:= \{D \mid D \text{ is an } \mathcal{L}\text{-concept description}\}, \\ M &:= \mathcal{C}, \\ I &:= \{(D, C) \mid C \sqsubseteq D\}. \end{aligned}$$

$\diamond$

As an easy consequence of the definition of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ and of the lcs, we obtain that the extent of a set $B \subseteq M$ is closely related to the lcs of this set:

Lemma 4.7.2. *Let B, B_1, B_2 be subsets of M.*

1. $B' = \{D \in G \mid \mathsf{lcs}(B) \sqsubseteq D\}$.
2. $B_1' \subseteq B_2'$ *iff* $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$.

Now, we can again show that implications correspond to subsumption relationships between the corresponding least common subsumers.

Lemma 4.7.3. *Let B_1, B_2 be subsets of M. The implication $B_1 \to B_2$ holds in $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ iff $\mathsf{lcs}(B_2) \sqsubseteq \mathsf{lcs}(B_1)$.*

As an immediate consequence of this lemma, the Duquenne-Guigues base $\mathcal{J}$ of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ yields a representation of all subsumption relationships of the form $\mathsf{lcs}(B_1) \sqsubseteq \mathsf{lcs}(B_2)$ for subsets B_1, B_2 of G. Given this base $\mathcal{J}$, any question of the form "$\mathsf{lcs}(B_1) \sqsubseteq \mathsf{lcs}(B_2)$?" can then be answered in time linear in the size of $\mathcal{J} \cup \{B_1 \to B_2\}$. Another easy consequence of the lemma is that the concept lattice of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ coincides with the inverse subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$. The proof of this fact is very similar to the proof of Theorem 4.6.7.

Theorem 4.7.4. *The concept lattice of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ is isomorphic to the inverse subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$.*

If we want to use attribute exploration to compute the concept lattice and the Duquenne-Guigues base, we need an expert for the context $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$. This expert must be able to answer the questions asked by the attribute exploration algorithm, i.e., given an implication $B_1 \to B_2$, it must be able to decide whether this implication holds in $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$. If the implication does not hold, it must be able to compute a counterexample, i.e., an object $g \in B_1' \setminus B_2'$.

If the language $\mathcal{L}$ is such that the lcs is computable and subsumption is decidable (which is, e.g., the case for $\mathcal{L} = \mathcal{EL}$), then we can implement such an expert.

Proposition 4.7.5. *Given a subsumption algorithm for $\mathcal{L}$ as well as an algorithm for computing the lcs of a finite set of $\mathcal{L}$-concept descriptions, these algorithms can be used to obtain an expert for the context $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$.*

Using this expert, an application of the attribute exploration algorithm yields

- all intents of formal concepts of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$, and thus the concept lattice of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$, which coincides with the inverse subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$ (by Theorem 4.7.4);
- the Duquenne-Guigues base of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$, which yields a compact representation of this hierarchy (by Lemma 4.7.3); and
- a finite subcontext of $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ that has the same concept intents as $\mathbb{K}_{\mathcal{L}}(\mathcal{C})$ and the same $(\cdot)''$ operation on sets of attributes.

Using the output of attribute exploration, one can then employ the usual tools for drawing concept lattices in order to present the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$ to the knowledge engineer.

4.8 The hierarchy of infima of a partially ordered set

The results presented in Sections 4.6 and 4.7 are very similar, and the proofs are based on rather generic arguments (i.e., they use almost no specific properties of the lcs or the conjunctions of defined concepts). Thus, one may ask whether the constructions and arguments used there can be generalized.

Consider a partially ordered set $(P, \preceq)$ for which all finite infima exist, i.e., if B is a finite subset of P, then there exists an element $\wedge B \in P$ that is the greatest element of P smaller than all elements of B. From the algorithmic point of view we assume that there are algorithms for deciding the relation $\preceq$ and for computing $\wedge B$ for all finite subsets B of P. In Section 4.6, P is the set of all concept descriptions of the DL under consideration, $\preceq$ is subsumption w.r.t. the TBox ($\sqsubseteq_{\mathcal{T}}$), and the infimum of a finite set of such descriptions is their conjunction. To be more precise, P is the set of all equivalence classes $[C]_{\equiv}$ of concept descriptions C, and the partial order is the partial order $\sqsubseteq_{\equiv}$ induced by subsumption w.r.t. $\mathcal{T}$ on these equivalence classes. In Section 4.7, P is again the set of all concept descriptions of the DL under consideration, $\preceq$ is inverse subsumption ($\sqsupseteq$) and the infimum is given by the lcs. Since we take inverse subsumption, the lcs, which is the supremum w.r.t. subsumption, is indeed the infimum.

Definition 4.8.1 (Infimum closure). Given a finite subset N of P, its *infimum closure* is the set

$$\overline{\wedge}(N) := \{\wedge B \mid B \subseteq N\}.$$

◇

In Section 4.6, N is the set of all concept names defined in the TBox, and $\overline{\wedge}(N)$ is the set of all conjunctions of such names. In Section 4.7, N is a finite set $\mathcal{C}$ of concept descriptions, and $\overline{\wedge}(N)$ is the set of all least common subsumers of subsets of $\mathcal{C}$.

Since N is finite, $(\overline{\wedge}(N), \preceq)$ is a complete semilattice, and thus a complete lattice. We are interested in computing this lattice. In Section 4.6, $(\overline{\wedge}(N), \preceq)$ is the hierarchy of all conjunctions of defined concepts, and in Section 4.7 $(\overline{\wedge}(N), \preceq)$ is the hierarchy of all least common subsumers of subsets of $\mathcal{C}$. In order to compute $(\overline{\wedge}(N), \preceq)$, we define a formal context with attribute set N such that the concept lattice of this context is isomorphic to $(\overline{\wedge}(N), \preceq)$.

Definition 4.8.2. Let $(P, \preceq)$ be a partially ordered set for which all finite infima exist, and let N be a finite subset of P. The formal context $\mathbb{K}_{\preceq}(N) = (G, M, I)$ is defined as follows:

$$\begin{aligned} G &:= P, \\ M &:= N, \\ I &:= \{(g, m) \mid g \preceq m \ \text{ for } g \in G,\ m \in M\}. \end{aligned}$$

◇

Valid implications in $\mathbb{K}_{\preceq}(N)$ correspond to $\preceq$-relationships between infima of subsets of N. The proof of this result is an easy generalization of the proof of Lemma 4.6.6.

Lemma 4.8.3. *Let B_1, B_2 be subsets of $M = N$. The implication $B_1 \to B_2$ holds in $\mathbb{K}_{\preceq}(N)$ iff $\wedge B_1 \preceq \wedge B_2$.*

Proof. First, we prove the *only if* direction. If the implication $B_1 \to B_2$ holds in $\mathbb{K}_{\preceq}(N)$, then this means that the following holds for all objects $g \in G$: if $g \preceq m$ holds for all $m \in B_1$, then $g \preceq m$ also holds for all $m \in B_2$. Thus, if we take $\wedge B_1$ as object g, we obviously have $\wedge B_1 \preceq n$ for all $n \in B_1$, and thus $\wedge B_1 \preceq n$ for all $n \in B_2$, which shows that $\wedge B_1 \preceq \wedge B_2$.

Second, we prove the *if* direction. If $\wedge B_1 \preceq \wedge B_2$, then any object g satisfying $g \preceq \wedge B_1$ also satisfies $g \preceq \wedge B_2$ by the transitivity of $\preceq$. Consequently, if $g \preceq m$ for all $m \in B_1$, then $g \preceq \wedge B_1 \preceq \wedge B_2 \preceq m$ for all $m \in B_2$, i.e., if g satisfies all attributes in B_1, it also satisfies all attributes in B_2. This shows that the implication $B_1 \to B_2$ holds in $\mathbb{K}_{\preceq}(N)$. □

The proof of the following theorem is an easy generalization of the proof of Theorem 4.6.3 and Theorem 4.6.7.

Theorem 4.8.4. *The concept lattice of the context* $\mathbb{K}_{\preceq}(N)$ *is isomorphic to the lattice* $(\overline{\wedge}(N), \preceq)$.

Proof. We define a mapping π from the formal concepts of the context $\mathbb{K}_{\preceq}(N)$ to $\overline{\wedge}(N)$ as follows:

$$\pi(A, B) = \wedge B.$$

Since B is a finite subset of M and $M = N$, the definition of $\overline{\wedge}(N)$ implies that $\wedge B \in \overline{\wedge}(N)$.

For formal concepts $(A_1, B_1), (A_2, B_2)$ of $\mathbb{K}_{\preceq}(N)$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $B_2 \subseteq B_1$. Since B_1 is an intent, we have $B_1 = B_1''$, and thus $B_2 \subseteq B_1$ iff $B_2 \subseteq B_1''$ iff the implication $B_1 \to B_2$ holds in $\mathbb{K}_{\preceq}(N)$ iff $\wedge B_1 \preceq \wedge B_2$. Overall, we have thus shown that π is an order embedding (and thus injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $\wedge B_1 \preceq \wedge B_2$.

It remains to be shown that π is surjective as well. Let B be an arbitrary subset of $M = N$. We must show that $\wedge B$ can be obtained as an image under the mapping π. We know that (B', B'') is a formal concept of $\mathbb{K}_{\preceq}(N)$, and thus it is sufficient to show that $\pi(B', B'') = \wedge B$, i.e., $\wedge B'' = \wedge B$. Obviously, $B \subseteq B''$ implies $\wedge B'' \preceq \wedge B$. Conversely, the implication $B \to B''$ holds in $\mathbb{K}_{\preceq}(N)$, and thus Lemma 4.8.3 yields $\wedge B \preceq \wedge B''$. Since $\preceq$ is antisymmetric, this shows $\wedge B = \wedge B''$. □

If we want to use attribute exploration to compute the concept lattice and the Duquenne-Guigues base of $\mathbb{K}_{\preceq}(N)$, we need an expert for this context. The proof of the next proposition is a generalization of the proofs of Proposition 4.6.4 and of Proposition 4.6.8.

Proposition 4.8.5. *Given a decision procedure for* $\preceq$ *as well as an algorithm for computing the infima of all finite subsets of* M*, these algorithms can be used to obtain an expert for the context* $\mathbb{K}_{\preceq}(N)$.

Proof. First, we show how to decide whether a given implication $B_1 \to B_2$ holds in $\mathbb{K}_{\preceq}(N)$ or not. By Lemma 4.8.3, we know that $B_1 \to B_2$ holds in $\mathbb{K}_{\preceq}(N)$ iff $\wedge B_1 \preceq \wedge B_2$. Obviously, $\wedge B_1 \preceq \wedge B_2$ iff $\wedge B_1 \preceq m$ for all $m \in B_2$. Thus, to answer the question "$B_1 \to B_2$?", we first compute $\wedge B_1$ and then use the decision procedure for $\preceq$ to test whether $\wedge B_1 \preceq m$ holds for all $m \in B_2$.

Second, assume that $B_1 \to B_2$ does not hold in $\mathbb{K}_{\preceq}(N)$, i.e., $\wedge B_1 \not\preceq \wedge B_2$. We claim that $\wedge B_1$ is a counterexample, i.e., $\wedge B_1 \in B_1'$ and $\wedge B_1 \notin B_2'$. This is an immediate consequence of the facts that $B_i' = \{g \in G \mid g \preceq$

m for all $m \in B_i\} = \{g \in G \mid g \preceq \wedge B_i\}$ $(i = 1, 2)$ and that $\wedge B_1 \preceq \wedge B_1$ and $\wedge B_1 \not\preceq \wedge B_2$.

What the attribute exploration algorithm (Algorithm 2 in Section 3.4) really needs is the row corresponding to this object in the matrix corresponding to I. This row can easily be computed using the decision procedure for $\preceq$: for each $m \in M = N$, we use this decision procedure to test whether $\wedge B_1 \preceq m$ holds or not. □

To sum up, we have shown that attribute exploration can be used to compute (a representation of) the lattice $(\bar{\wedge}(N), \preceq)$ provided that $\preceq$ is decidable and all finite infima are computable. The results presented in Section 4.6 and in Section 4.7 are instances of this general result. The fact that the approach described in Section 4.7 (and first presented in [BM00]) can be generalized in this direction has already been mentioned in [GK01] (Section 3), but not worked out in detail.

Chapter 5

Completing DL Knowledge Bases

In the present chapter we present an approach for extending both the terminological and the assertional part of a Description Logic knowledge base by using information provided by the knowledge base and by a domain expert. Our approach provides a basis for formally well-founded techniques and tools that support the knowledge engineer in checking whether a knowledge base contains all the relevant information about the application domain, and in extending the knowledge base appropriately if this is not the case. The use of techniques from Formal Concept Analysis ensures that, on the one hand, the interaction with the expert is kept to a minimum, and, on the other hand, it enables us to show that the extended knowledge base is complete in a certain, well-defined sense. We start the chapter with Section 5.1, where we give an informal definition of the notion of completeness of a knowledge base w.r.t. a fixed model. In Section 5.2 we introduce our variant of Formal Concept Analysis that can deal with incomplete knowledge represented in the form of a so-called partial context, and describe an extension of the attribute exploration method that works with partial contexts. In Section 5.3 we show how a Description Logic knowledge base gives rise to a partial context, formally define the notion of a completion of a knowledge base w.r.t. a fixed model, and show that the attribute exploration algorithm we have developed in the previous section can be used to complete a knowledge base. We conclude the chapter with Section 5.4, where we describe our implementation of the method.

5.1 Completeness of a DL knowledge base

Description Logics are employed in various application domains, such as natural language processing, configuration, databases, and bio-medical ontologies, but their most notable success so far is due to the fact that DLs provide the logical underpinning of OWL [HPSvH03], the standard ontology

language for the semantic web [BLHL01]. As a consequence of this standardization, several ontology editors [KFNM04, OVSM04, KPS+06] support OWL, and ontologies written in OWL are employed in more and more applications. As the size of these ontologies grows, tools that support improving their quality become more important. The tools available until now use DL reasoning to detect inconsistencies and to infer consequences, i.e., implicit knowledge that can be deduced from the explicitly represented knowledge. There are also promising approaches that allow to pinpoint the reasons for inconsistencies and for certain consequences, and that help the ontology engineer to resolve inconsistencies and to remove unwanted consequences [SC03, KPSG06]. These approaches address the quality dimension of *soundness* of an ontology, both within itself (consistency) and w.r.t. the intended application domain (no unwanted consequences). In this chapter we are concerned with a different quality dimension: *completeness*. We provide a basis for formally well-founded techniques and tools that support the ontology engineer in checking whether an ontology contains all the relevant information about the application domain, and to extend the ontology appropriately if this is not the case.

As already mentioned in Chapter 2, a DL knowledge base (nowadays often called ontology) usually consists of two parts, the terminological part (TBox), which defines concepts and also states additional constraints (so-called general concept inclusions, or GCIs for short) on the interpretation of these concepts, and the assertional part (ABox), which describes individuals and their relationship to each other and to concepts. Given an application domain and a DL knowledge base describing it, we can ask whether the knowledge base contains all the relevant information[1] about the domain:

- Are all the relevant constraints that hold between concepts in the domain captured by the TBox?
- Are all the relevant individuals existing in the domain present in the ABox?

As an example, consider the OWL ontology for human protein phosphatases that has been described and used in [WBH+05]. This ontology was developed based on information from peer-reviewed publications. The human protein phosphatase family has been well characterised experimentally, and detailed knowledge about different classes of such proteins is available. This knowledge is represented in the terminological part of the ontology. Moreover, a large set of human phosphatases has been identified and documented by expert biologists. These are described as individuals in the assertional part of the ontology. One can now ask whether the information about protein phosphatases contained in this ontology is complete: Are

[1] The notion of "relevant information" must, of course, be formalized appropriately for this problem to be addressed algorithmically.

all the relationships that hold among the introduced classes of phosphatases captured by the constraints in the TBox, or are there relationships that hold in the domain, but do not follow from the TBox? Are all possible kinds of human protein phosphatases represented by individuals in the ABox, or are there phosphatases that have not yet been included in the ontology or even not yet been identified?

Such questions cannot be answered by an automated tool alone. Clearly, to check whether a given relationship between concepts—which does not follow from the TBox—holds in the domain, one needs to ask a domain expert, and the same is true for questions regarding the existence of individuals not described in the ABox. The rôle of the automated tool is to ensure that the expert is asked as few questions as possible; in particular, she should not be asked trivial questions, i.e., questions that could actually be answered based on the represented knowledge. In the above example, answering a non-trivial question regarding human protein phosphatases may require the biologist to study the relevant literature, query existing protein databases, or even to carry out new experiments. Thus, the expert may be prompted to acquire new biological knowledge.

The attribute exploration method we have described in Section 3.4 has proved to be a successful knowledge acquisition method in various application domains. One of the earliest applications of this approach is described in [Wil82], where the domain is lattice theory, and the goal of the exploration process is to find, on the one hand, all valid relationships between properties of lattices (like being distributive), and, on the other hand, to find counterexamples to all the relationships that do not hold. To answer a query whether a certain relationship holds, the lattice theory expert must either confirm the relationship (by using results from the literature or carrying out a new proof for this fact), or give a counterexample (again, by either finding one in the literature or constructing a new one).

Although this sounds very similar to what is needed in our case, we cannot directly use this approach. The main reason is the open-world semantics of description logic knowledge bases. Consider an individual i from an ABox $\mathcal{A}$ and a concept C occurring in a TBox $\mathcal{T}$. If we cannot deduce from $\mathcal{T}$ and $\mathcal{A}$ that i is an instance of C, then we do not assume that i does not belong to C. Instead, we only accept this as a consequence if $\mathcal{T}$ and $\mathcal{A}$ imply that i is an instance of $\neg C$. Thus, our knowledge about the relationships between individuals and concepts is incomplete: if $\mathcal{T}$ and $\mathcal{A}$ imply neither $C(i)$ nor $\neg C(i)$, then we do not know the relationship between i and C. In contrast, classical FCA and attribute exploration assume that the knowledge about objects is complete: a cross in row g and column m of a formal context says that object g has attribute m, and the absence of a cross is interpreted as saying that g does not have m.

There has been some work on how to extend FCA and attribute exploration from complete knowledge to the case of partial knowledge [Gan99,

BH00, Hol04a, Hol04b, BH05, Rud06], and how to evaluate formulas in formal contexts that do not contain complete information [Obi02]. However, this work is based on assumptions that are different from ours. In particular, they assume that the expert cannot answer all queries and, as a consequence, the knowledge obtained after the exploration process may still be incomplete and the relationships between concepts that are produced in the end fall into two categories: relationships that are valid no matter how the incomplete part of the knowledge is completed, and relationships that are valid only in some completions of the incomplete part of the knowledge. In contrast, our intention is to complete the knowledge base, i.e., in the end we want to have complete knowledge about these relationships. What may be incomplete is the description of individuals used during the exploration process.

In the next section, we introduce our variant of FCA that can deal with partial contexts, and describe an attribute exploration procedure that works with partial contexts. In Section 5.3, we show how a DL knowledge base gives rise to a partial context, define the notion of a completion of a knowledge base w.r.t. a fixed model, and show that the attribute exploration algorithm developed in the previous section can be used to complete a knowledge base. The results presented in this chapter have been published in [BGSS06], [BGSS07b], and [BGSS07a].

5.2 An extension of FCA for completing DL knowledge bases

In this section, we extend the classical approach to FCA described in Chapter 3 to the case of objects that have only a partial description in the sense that, for some attributes, it is not known whether they are satisfied by the object or not. We need this extension in order to deal with the open-world semantics of the description logic knowledge bases, and explore them using attribute exploration.

5.2.1 Partial contexts

In our extension, we represent partial knowledge in the form of partial object descriptions and a partial context, which is a set of partial object descriptions. We start with formal definitions of these notions.

Definition 5.2.1 (Partial object description). A *partial object description (pod)* is a tuple (A, S) where $A, S \subseteq M$ are such that $A \cap S = \emptyset$. We call such a pod a *full object description (fod)* if $A \cup S = M$. A set of pods is called a *partial context* and a set of fods a *full context* . ◇

Note that the notion of a full context introduced in this definition coincides with the notion of a formal context: a set of fods $\overline{\mathcal{K}}$ corresponds

to the formal context $\mathbb{K}_{\overline{\mathcal{K}}} := (\overline{\mathcal{K}}, M, I)$, where $(\overline{A}, \overline{S})Im$ iff $m \in \overline{A}$ for all $(\overline{A}, \overline{S}) \in \overline{\mathcal{K}}$.

A partial context can be extended by either adding new pods or by extending existing pods.

Definition 5.2.2 (Realizer). We say that the pod (A', S') *extends* the pod (A, S), and write this as $(A, S) \leq (A', S')$, if $A \subseteq A'$ and $S \subseteq S'$. Similarly, we say that the partial context $\mathcal{K}'$ *extends* the partial context $\mathcal{K}$, and write this as $\mathcal{K} \leq \mathcal{K}'$, if every pod in $\mathcal{K}$ is extended by some pod in $\mathcal{K}'$. If $\overline{\mathcal{K}}$ is a full context and $\mathcal{K} \leq \overline{\mathcal{K}}$, then $\overline{\mathcal{K}}$ is called a *realizer* of $\mathcal{K}$. If $(\overline{A}, \overline{S})$ is a fod and $(A, S) \leq (\overline{A}, \overline{S})$, then we also say that $(\overline{A}, \overline{S})$ *realizes* (A, S). ◇

5.2.2 Implications in partial contexts

Next, we extend the definition of the implications of a formal context to the case of partial contexts.

Definition 5.2.3 (Implication in partial contexts). Let $L, R \subseteq M$. The implication $L \to R$ is *refuted* by the pod (A, S) if $L \subseteq A$ and $R \cap S \neq \emptyset$. It is *refuted* by the partial context $\mathcal{K}$ if it is refuted by at least one element of $\mathcal{K}$. The set of implications that are not refuted by a given partial context $\mathcal{K}$ is denoted by $Imp(\mathcal{K})$. The set of all fods that do not refute a given set of implications $\mathcal{L}$ is denoted by $Mod(\mathcal{L})$. ◇

If (A, S) is a fod and $L \to R$ an implication, then (A, S) does not refute $L \to R$ iff $L \subseteq A$ implies $R \cap S = \emptyset$ iff $L \subseteq A$ implies $R \subseteq M \setminus S = A$. Thus, the implication $L \to R$ is not refuted by the full context $\overline{\mathcal{K}}$ iff it holds in the corresponding formal context $\mathbb{K}_{\overline{\mathcal{K}}}$.

The following simple facts regarding the connection between $Imp(\cdot)$, $Mod(\cdot)$, and the consequence operator for implications will be employed later on without explicitly mentioning their application:

- If $\overline{\mathcal{K}}$ is a full context and $\mathcal{L}$ a set of implications, then $\overline{\mathcal{K}} \subseteq Mod(\mathcal{L})$ iff $\mathcal{L} \subseteq Imp(\overline{\mathcal{K}})$.
- If $\mathcal{K}$ is a partial context and $\mathcal{L}$ a set of implications, then $\mathcal{L} \subseteq Imp(\mathcal{K})$ implies that every implication that follows from $\mathcal{L}$ belongs to $Imp(\mathcal{K})$.

The following is a trivial fact regarding the connection between partial contexts and the implications they do not refute.

Proposition 5.2.4. *For a given set $P \subseteq M$ and a partial context $\mathcal{K}$,*

$$\mathcal{K}(P) := M \setminus \bigcup \{S \mid (A, S) \in \mathcal{K}, P \subseteq A\}$$

is the largest subset of M such that $P \to \mathcal{K}(P)$ is not refuted by $\mathcal{K}$.

The following facts are immediate consequences of the definition of $\mathcal{K}(\cdot)$:

- If $P \subseteq Q$, then $\mathcal{K}(P) \subseteq \mathcal{K}(Q)$.
- If $\mathcal{K} \leq \mathcal{K}'$, then $\mathcal{K}'(P) \subseteq \mathcal{K}(P)$.

For a full context $\overline{\mathcal{K}}$, the operator $\overline{\mathcal{K}}(\cdot)$ coincides with the $\cdot''$ operator of the corresponding formal context $\mathbb{K}_{\overline{\mathcal{K}}}$. In fact, if $\mathcal{L}$ is a base for $Imp(\mathbb{K}_{\overline{\mathcal{K}}})$, then we have $m \in P''$ iff $m \in \mathcal{L}(P)$ iff $P \to \{m\}$ follows from $\mathcal{L}$ iff $P \to \{m\}$ holds in $\mathbb{K}_{\overline{\mathcal{K}}}$ iff $P \to \{m\}$ is not refuted by $\overline{\mathcal{K}}$ iff $m \in \overline{\mathcal{K}}(P)$. Note that if $\mathcal{K}$ is not a full context, then $\mathcal{K}(\cdot)$ will not be a closure operator since it will not be idempotent.

The following proposition shows the relation between refutation by a partial context and refutation by the realizers of this partial context.

Proposition 5.2.5. *Let $\mathcal{K}$ be a partial context. An implication is refuted by $\mathcal{K}$ iff it is refuted by all realizers of $\mathcal{K}$.*

Proof. First, let $L, R \subseteq M$ be such that $L \to R$ is refuted by $\mathcal{K}$, and let $\overline{\mathcal{K}}$ be a realizer of $\mathcal{K}$. Then, by the definition of refutation, there is an $(A, S) \in \mathcal{K}$ such that $L \subseteq A$ and $R \cap S \neq \emptyset$, and by the definition of a realizer, there is a fod $(\overline{A}, \overline{S}) \in \overline{\mathcal{K}}$ such that $A \subseteq \overline{A}$ and $S \subseteq \overline{S}$. Obviously, we have $L \subseteq \overline{A}$ and $R \cap \overline{S} \neq \emptyset$. Thus, $L \to R$ is refuted by $\overline{\mathcal{K}}$ as well.

Second, assume the implication $L \to R$ is not refuted by $\mathcal{K}$, i.e., for every pod $(A, S) \in \mathcal{K}$ we have that $L \subseteq A$ implies $R \cap S = \emptyset$. We define a realizer $\overline{\mathcal{K}}$ of $\mathcal{K}$ as follows. Consider a pod $(A, S) \in \mathcal{K}$. If $L \not\subseteq A$, then we add $(A, M \setminus A)$ to $\overline{\mathcal{K}}$: obviously, $(A, M \setminus A)$ realizes (A, S) and does not refute $L \to R$. If $L \subseteq A$, then we also have $R \cap S = \emptyset$, and we add $(M \setminus S, S)$ to $\overline{\mathcal{K}}$: obviously, $(M \setminus S, S)$ realizes (A, S) and does not refute $L \to R$. □

Note that the if-direction of this proposition need not hold if we consider a set of implications rather than a single implication. For example, consider the implications $\{a, b\} \to \{c\}, \{a\} \to \{b\}$. The partial context that consists of the single pod $(\{a\}, \{c\})$ does not refute any of these two implications, but each realizer of this partial context refutes one of them.

In the proof of the only-if-direction, we did not make use of the fact that $\overline{\mathcal{K}}$ is a full context. Thus, this direction also holds for partial contexts.

Lemma 5.2.6. *If $\mathcal{K}, \mathcal{K}'$ are partial contexts such that $\mathcal{K} \leq \mathcal{K}'$, then every implication refuted by $\mathcal{K}$ is also refuted by $\mathcal{K}'$.*

5.2.3 Undecided implications

In contrast to existing work on extending FCA to the case of partial knowledge [BH00, Hol04a, Hol04b, BH05], we do *not* assume that the expert has only partial knowledge and thus cannot answer all implication questions. In principle, our expert is assumed to have access to a full context $\overline{\mathcal{K}}$ and thus can answer all implication questions w.r.t. $\overline{\mathcal{K}}$, though finding these answers

may involve extensive literature study, or even proving new mathematical theorems or carrying out new experiments, etc. What is partial is the subcontext that the attribute exploration algorithm works with. The reason is that the initial context may be partial, and the same is true for the counterexamples that the experts provides for implications that do not hold in $\overline{\mathcal{K}}$.

More formally, we consider the following setting. We are given an initial (possibly empty) partial context $\mathcal{K}$, an initially empty set of implications $\mathcal{L}$, and a full context $\overline{\mathcal{K}}$ that is a realizer of $\mathcal{K}$. The expert answers implication questions "$L \to R$?" w.r.t. the full context $\overline{\mathcal{K}}$. More precisely, if the answer is "yes," then $\overline{\mathcal{K}}$ does not refute $L \to R$ (and thus $L \to R$ holds in the corresponding formal context $\mathbb{K}_{\overline{\mathcal{K}}}$). The implication $L \to R$ is then added to $\mathcal{L}$. Otherwise, the expert extends the current context $\mathcal{K}$ such that the extended context refutes $L \to R$ and still has $\overline{\mathcal{K}}$ as a realizer. Consequently, the following invariant will be satisfied by $\mathcal{K}, \overline{\mathcal{K}}, \mathcal{L}$:

$$\mathcal{K} \leq \overline{\mathcal{K}} \subseteq \mathit{Mod}(\mathcal{L}).$$

Our aim is to enrich $\mathcal{K}$ and $\mathcal{L}$ such that

- eventually $\mathcal{L}$ is not only sound, but also complete for $\mathit{Imp}(\overline{\mathcal{K}})$,
- and $\mathcal{K}$ refutes all other implications (i.e., all the implications refuted by $\overline{\mathcal{K}}$).

As in the classical case, we want to do this by asking as few questions as possible to the expert.

Definition 5.2.7. Let $\mathcal{L}$ be a set of implications and $\mathcal{K}$ a partial context. An implication is called *undecided* w.r.t. $\mathcal{K}$ and $\mathcal{L}$ if it neither follows from $\mathcal{L}$ nor is refuted by $\mathcal{K}$. It is *decided* w.r.t. $\mathcal{K}$ and $\mathcal{L}$ if it is not undecided w.r.t. $\mathcal{K}$ and $\mathcal{L}$. ◇

In principle, our attribute exploration algorithm tries to decide all undecided implications by either adding the implication to $\mathcal{L}$ or extending $\mathcal{K}$ such that it refutes the implication. If all implications are decided, then our goal is achieved.

Proposition 5.2.8. *Assume that $\mathcal{K} \leq \overline{\mathcal{K}} \subseteq \mathit{Mod}(\mathcal{L})$ and that all implications are decided w.r.t. $\mathcal{K}$ and $\mathcal{L}$. Then $\mathcal{L}$ is complete for $\mathit{Imp}(\overline{\mathcal{K}})$ and $\mathcal{K}$ refutes all implications not belonging to $\mathit{Imp}(\overline{\mathcal{K}})$.*

Proof. First, assume that there is an implication $L \to R$ in $\mathit{Imp}(\overline{\mathcal{K}})$ that does not follow from $\mathcal{L}$. By our assumption, $L \to R$ is decided w.r.t. $\mathcal{K}$ and $\mathcal{L}$, and thus it is refuted by $\mathcal{K}$. However, according to Proposition 5.2.5, it is then also refuted by the realizer $\overline{\mathcal{K}}$ of $\mathcal{K}$, which contradicts our assumption that $L \to R$ belongs to $\mathit{Imp}(\overline{\mathcal{K}})$.

Second, assume that $L \to R$ is an implication that is refuted by $\overline{\mathcal{K}}$, but is not refuted by $\mathcal{K}$. Since $L \to R$ is decided, this implies that $L \to R$ follows from $\mathcal{L}$. However, $\overline{\mathcal{K}} \subseteq \mathit{Mod}(\mathcal{L})$ implies $\mathcal{L} \subseteq \mathit{Imp}(\overline{\mathcal{K}})$, and thus $L \to R$ also belongs to $\mathit{Imp}(\overline{\mathcal{K}})$. This contradicts our assumption that $L \to R$ is refuted by $\overline{\mathcal{K}}$. □

How can we find the undecided implications? The following proposition motivates why it is sufficient to consider implications whose left-hand sides are $\mathcal{L}$-closed. It is an immediate consequence of the fact that $\mathcal{L}(\cdot)$ is a closure operator, and thus idempotent.

Proposition 5.2.9. *Let $\mathcal{L}$ be a set of implications and $L \to R$ an implication. Then, $L \to R$ follows from $\mathcal{L}$ iff $\mathcal{L}(L) \to R$ follows from $\mathcal{L}$.*

Given an $\mathcal{L}$-closed set L as left-hand side, what kind of right-hand sides should we consider? Obviously, we need not consider right-hand sides R for which the implication $L \to R$ is refuted by $\mathcal{K}$: such implications are already decided. By Proposition 5.2.4, the largest right-hand side R such that $L \to R$ is not refuted by $\mathcal{K}$ is $R = \mathcal{K}(L)$. It is actually enough to consider just this right-hand side. In fact, once we have decided $L \to \mathcal{K}(L)$ (by either extending $\mathcal{K}$ such that it refutes the implication or adding the implication to $\mathcal{L}$), all implications $L \to R'$ with $R' \subseteq \mathcal{K}(L)$ are also decided.

In order to enumerate all left-hand sides, we again use the lectic order (see Definition 3.2.6) and the procedure derived from Proposition 3.2.7 for enumerating all $\mathcal{L}$-closed sets w.r.t. this order.

5.2.4 Attribute exploration in partial contexts

Until now, we have talked as if there was a fixed set of implications $\mathcal{L}$ and a fixed partial context $\mathcal{K}$ to work with. As it is also the case in the classical attribute exploration algorithm, both $\mathcal{L}$ and $\mathcal{K}$ change during the run of our procedure. We start with an empty set of implications and an initial partial context, and the procedure can extend both. The following proposition shows that the left-hand sides of the previously added implications are also closed with respect to the extended set of implications. This is due to the fact that the left-hand sides are enumerated in lectic order.

Proposition 5.2.10. *Let $\mathcal{L}$ be a set of implications and $P_1 < \ldots < P_n$ the lectically first n $\mathcal{L}$-closed sets. If $\mathcal{L}$ is extended with $L \to R$ s.t. L is $\mathcal{L}$-closed and $P_n < L$, then $P_1, \ldots, P_n$ are still the lectically first n closed sets with respect to the extended set of implications.*

Proof. If $P_1 < \ldots < P_n$ and $P_n < L$, then $P_i < L$ for $i = 1, \ldots, n$ by transitivity of $<$. Since $<$ is irreflexive and contains the strict subset order, $L \not\subseteq P_i$ holds for $i = 1, \ldots, n$. Consequently, the $\mathcal{L}$-closed sets P_i are closed

w.r.t. $L \to R$, and thus also w.r.t. the extended set of implications $\mathcal{L}' := \mathcal{L} \cup \{L \to R\}$.

It remains to show that $P_1, \ldots, P_{n-1}$ are all the $\mathcal{L}'$-closed sets smaller than P_n. Thus, assume that $P < P_n$ is an $\mathcal{L}'$-closed set. Since $\mathcal{L} \subseteq \mathcal{L}'$, we know that P is also $\mathcal{L}$-closed, and thus it is actually one of the sets P_i, $1 \leq i < n$. □

If an implication has been added because the expert has stated that it holds in $\overline{\mathcal{K}}$, then we can extend the current context $\mathcal{K}$ by applying the implications to the first component of every pod in $\mathcal{K}$. To be more precise, for a partial context $\mathcal{K}$ and a set of implications $\mathcal{L}$ we define

$$\mathcal{L}(\mathcal{K}) := \{(\mathcal{L}(A), S) \mid (A, S) \in \mathcal{K}\}.$$

The following is a simple consequence of this definition.

Proposition 5.2.11. *Let $\mathcal{K} \leq \overline{\mathcal{K}}$ be a partial and a full context, respectively, and let $\mathcal{L}$ be a set of implications such that $\mathcal{L} \subseteq Imp(\overline{\mathcal{K}})$. Then $\mathcal{L}(\mathcal{K})$ is a partial context and $\mathcal{K} \leq \mathcal{L}(\mathcal{K}) \leq \overline{\mathcal{K}}$.*

Proof. Obviously, $\mathcal{K} \leq \mathcal{L}(\mathcal{K})$ follows from the fact that $A \subseteq \mathcal{L}(A)$. To show $\mathcal{L}(\mathcal{K}) \leq \overline{\mathcal{K}}$, we consider a pod $(A, S) \in \mathcal{K}$. We must show that $(\mathcal{L}(A), S)$ is realized by some fod in $\overline{\mathcal{K}}$. We know that (A, S) is realized by some fod in $\overline{\mathcal{K}}$, i.e., there is a fod $(\overline{A}, \overline{S}) \in \overline{\mathcal{K}}$ such that $A \subseteq \overline{A}$ and $S \subseteq \overline{S}$. Since $\mathcal{L} \subseteq Imp(\overline{\mathcal{K}})$, we have $\mathcal{L}(\overline{A}) = \overline{A}$, and thus $\mathcal{L}(A) \subseteq \mathcal{L}(\overline{A}) = \overline{A}$. This shows that $(\overline{A}, \overline{S})$ also realizes $(\mathcal{L}(A), S)$.

The fact that $\mathcal{L}(\mathcal{K})$ is a partial context, i.e., that $\mathcal{L}(A) \cap S = \emptyset$ holds for all $(A, S) \in \mathcal{K}$, is an immediate consequence of $\mathcal{L}(\mathcal{K}) \leq \overline{\mathcal{K}}$. □

Going from $\mathcal{K}$ to $\mathcal{L}(\mathcal{K})$ is actually only one way to extend the current context based on the already computed implications. For example, if we have the pod $(\{\ell\}, \{n\})$ and the implication $\{\ell, m\} \to \{n\}$ is not refuted by $\overline{\mathcal{K}}$, then we know that m must belong to the second component of every fod realizing $(\{\ell\}, \{n\})$. Consequently, we can extend $(\{\ell\}, \{n\})$ to $(\{\ell\}, \{m, n\})$. To allow also for this and possible other ways of extending the partial context, the formulation of the algorithm just says that, in case an implication is added, the partial context can also be extended.

Whenever an implication is not accepted by the expert, $\mathcal{K}$ will be extended to a context that refutes the implication and still has $\overline{\mathcal{K}}$ as a realizer. The following proposition shows that the right-hand sides of implications accepted by the expert and computed with respect to the smaller partial context are identical to the ones that would have been computed with respect to the extended one.

Proposition 5.2.12. *Let $\mathcal{K} \leq \mathcal{K}' \leq \overline{\mathcal{K}}$, where $\mathcal{K}, \mathcal{K}'$ are partial contexts and $\overline{\mathcal{K}}$ is a full context. If $L \to \mathcal{K}(L)$ is an implication that is not refuted by $\overline{\mathcal{K}}$, then $L \to \mathcal{K}(L)$ is not refuted by $\mathcal{K}'$ and $\mathcal{K}(L) = \mathcal{K}'(L)$.*

Algorithm 4 Attribute exploration for partial contexts

Initialization
$\mathcal{K}_0$ {initial partial context, realized by the underlying full context $\overline{\mathcal{K}}$}
$\mathcal{L}_0 := \emptyset$ {initial empty set of implications}
$P_0 := \emptyset$ {lectically smallest $\mathcal{L}_0$-closed subset of M}
$i := 0$
while $P_i \neq M$ **do**
  Compute $\mathcal{K}_i(P_i)$
  if $P_i \neq \mathcal{K}_i(P_i)$ **then** {$P_i \to \mathcal{K}_i(P_i)$ is undecided}
    Ask the expert if the undecided implication $P_i \to \mathcal{K}_i(P_i)$ is refuted by $\overline{\mathcal{K}}$
    if no **then** {$P_i \to \mathcal{K}_i(P_i)$ not refuted}
      $\mathcal{K}_{i+1} := \mathcal{K}'$ where $\mathcal{K}'$ is a partial context such that $\mathcal{K}_i \leq \mathcal{K}' \leq \overline{\mathcal{K}}$
      $\mathcal{L}_{i+1} := \mathcal{L}_i \cup \{P_i \to \mathcal{K}_i(P_i) \setminus P_i\}$
      $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P_i <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
    else {$P_i \to \mathcal{K}_i(P_i)$ refuted}
      Get a partial context $\mathcal{K}'$ from the expert such that $\mathcal{K}_i \leq \mathcal{K}' \leq \overline{\mathcal{K}}$ and $P_i \to \mathcal{K}_i(P_i)$ is refuted by $\mathcal{K}'$
      $\mathcal{K}_{i+1} := \mathcal{K}'$
      $P_{i+1} := P_i$
      $\mathcal{L}_{i+1} := \mathcal{L}_i$
    end if
  else {trivial implication}
    $\mathcal{K}_{i+1} := \mathcal{K}_i$
    $\mathcal{L}_{i+1} := \mathcal{L}_i$
    $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P_i <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
  end if
  $i := i + 1$
end while

Proof. We have $\mathcal{K}' \leq \overline{\mathcal{K}}$, and thus Proposition 5.2.5 implies that $L \to \mathcal{K}(L)$ is not refuted by $\mathcal{K}'$. Since $\mathcal{K} \leq \mathcal{K}'$, we have $\mathcal{K}'(L) \subseteq \mathcal{K}(L)$. If this inclusion were strict, then $L \to \mathcal{K}(L)$ would be refuted by $\mathcal{K}'$ by Proposition 5.2.4. Thus, we have shown that $\mathcal{K}(L) = \mathcal{K}'(L)$. □

Based on these considerations, our attribute exploration algorithm for partial contexts is described in Algorithm 4. Proposition 5.2.13 shows that this algorithm always terminates, and that it is correct.

Proposition 5.2.13. *Let M be a finite set of attributes, and $\overline{\mathcal{K}}$ and $\mathcal{K}_0$ respectively a full and a partial context over the attributes in M such that $\mathcal{K}_0 \leq \overline{\mathcal{K}}$. Then Algorithm 4 terminates, and upon termination it outputs a*

partial context $\mathcal{K}$ and a set of implications $\mathcal{L}$ such that

- *$\mathcal{L}$ is sound and complete for $Imp(\overline{\mathcal{K}})$, and*
- *$\mathcal{K}$ refutes every implication that is refuted by $\overline{\mathcal{K}}$.*

Proof. First, we show termination. The algorithm starts with the lectically smallest $\mathcal{L}_0$-closed set $P_0 = \mathcal{L}_0(\emptyset)$. At each execution of the while loop, it performs one of the following operations:

1. it extends the current set of implications $\mathcal{L}_i$, and continues with the lectically next closed set P_{i+1} computed by using the extended set of implications $\mathcal{L}_{i+1}$ (lines 12,13 in Algorithm 4).
2. it extends the current context $\mathcal{K}_i$ to a context $\mathcal{K}_{i+1}$ that does not refute any of the implications in $\mathcal{L}_i$, and continues with $P_{i+1} := P_i$ (lines 16,17).
3. it continues with the lectically next closed set P_{i+1}, computed by using the current set of implications $\mathcal{L}_i$ (line 23).

Steps of the form 1 or 3 can be executed only finitely often. In fact, in each of these steps, a lectically larger set is generated. Since M is finite, there are only finitely many subsets of M, and thus every strictly ascending chain w.r.t. $<$ is obviously finite. In steps of the form 2, the algorithm continues with $P_{i+1} := P_i$, but extends $\mathcal{K}_i$ to a partial context $\mathcal{K}_{i+1}$ that refutes the implication $P_i \to \mathcal{K}_i(P_i)$. Consequently, $\mathcal{K}_{i+1}(P_i) \subsetneq \mathcal{K}_i(P_i)$. This shows that, for a fixed set P_i, steps of the form 2 can also be applied only finitely often. Thus, we have shown that termination is guaranteed.

Second, to show soundness of the output set of implications $\mathcal{L}$ for $Imp(\overline{\mathcal{K}})$, it is sufficient to note that the invariant $\mathcal{K}_i \leq \overline{\mathcal{K}} \subseteq Mod(\mathcal{L}_i)$ is preserved throughout the run of the algorithm. Consequently, we also have $\mathcal{K} \leq \overline{\mathcal{K}} \subseteq Mod(\mathcal{L})$. But then $\overline{\mathcal{K}} \subseteq Mod(\mathcal{L})$ implies $\mathcal{L} \subseteq Imp(\overline{\mathcal{K}})$, and thus soundness of $\mathcal{L}$ for $Imp(\overline{\mathcal{K}})$.

Third, since we have $\mathcal{K} \leq \overline{\mathcal{K}} \subseteq Mod(\mathcal{L})$, Proposition 5.2.8 shows that completeness of $\mathcal{L}$ for $Imp(\mathcal{K})$ as well as the fact that $\mathcal{K}$ refutes every implication that is refuted by $\overline{\mathcal{K}}$ follow as soon as we have shown that every implication is decided w.r.t. $\mathcal{K}$ and $\mathcal{L}$. To see this, consider the sets $P_0 = \mathcal{L}_0(\emptyset), P_1, \ldots, P_n = M$ generated during the run of the algorithm. We have $P_0 < P_1 < \ldots < P_n$, and iterated applications of Proposition 5.2.10 show that $P_0, P_1, \ldots, P_n$ are all the $\mathcal{L}$-closed subsets of M.

Now, assume that the implication $L \to R$ is undecided w.r.t. $\mathcal{K}$ and $\mathcal{L}$. Thus, $L \to R$ does not follow from $\mathcal{L}$ and is not refuted by $\mathcal{K}$. By Proposition 5.2.9, $\mathcal{L}(L) \to R$ also does not follow from $\mathcal{L}$. In addition, since $L \subseteq \mathcal{L}(L)$, it is also not refuted by $\mathcal{K}$. Since $\mathcal{L}(L)$ is $\mathcal{L}$-closed, there is an i such that $\mathcal{L}(L) = P_i$. During iteration i of the algorithm, the implication $P_i \to \mathcal{K}_i(P_i)$ is considered.

First, assume that this implication is not refuted by $\overline{\mathcal{K}}$. Then, $P_i \to \mathcal{K}_i(P_i)$ follows from $\mathcal{L}_{i+1}$, and thus also from its superset $\mathcal{L}$. However, the fact that $P_i \to R$ is not refuted by $\mathcal{K}$ implies that it is also not refuted by $\mathcal{K}_i$ since $\mathcal{K}_i \leq \mathcal{K}$ (Lemma 5.2.6). Thus $R \subseteq \mathcal{K}_i(P_i)$ by Proposition 5.2.4, and the fact that $P_i \to \mathcal{K}_i(P_i)$ follows from $\mathcal{L}$ implies that $P_i \to R$ follows from $\mathcal{L}$, which yields a contradiction.

Second, assume that $P_i \to \mathcal{K}_i(P_i)$ is refuted by $\overline{\mathcal{K}}$. Then, $\mathcal{K}_i$ is extended to a partial context $\mathcal{K}_{i+1}$ that refutes the implication $P_i \to \mathcal{K}_i(P_i)$. If $\mathcal{K}_{i+1}$ also refutes $P_i \to R$, then we are done since $\mathcal{K}_{i+1} \leq \mathcal{K}$ implies that also $\mathcal{K}$ refutes $P_i \to R$, and thus $\mathcal{K}$ refutes $L \to R$ because $L \subseteq P_i$. Otherwise, note that $P_{i+1} = P_i$ and $\mathcal{L}_{i+1} = \mathcal{L}_i$, and thus, in the next iteration, the expert gets the implication $P_i \to \mathcal{K}_{i+1}(P_i)$. By our assumption, $P_i \to R$ is not refuted by $\mathcal{K}_{i+1}$, and thus $R \subseteq K_{i+1}(P_i)$. In addition, we have $\mathcal{K}_{i+1}(P_i) \subsetneq \mathcal{K}_i(P_i)$ due to the fact that $\mathcal{K}_{i+1}$ refutes $P_i \to \mathcal{K}_i(P_i)$.

If $P_i \to \mathcal{K}_{i+1}(P_i)$ is not refuted by $\overline{\mathcal{K}}$, then we can continue as in the first case above, and derive that $P_i \to R$ follows from $\mathcal{L}$. Otherwise, we can continue as in the second case. However, because in this case the size of the right-hand side of the implication given to the expert strictly decreases, we cannot indefinitely get the second case. This shows that, eventually, the implication $L \to R$ will become decided w.r.t. some $\mathcal{K}_j$ and $\mathcal{L}_j$ for some $j \geq i+1$, which contradicts our assumption that it is undecided w.r.t. their extensions $\mathcal{K}$ and $\mathcal{L}$. □

We have shown that the implication set $\mathcal{L}$ produced by the algorithm is sound and complete for $Imp(\overline{\mathcal{K}})$. Next, we show that this set is actually the Duquenne-Guigues base of $\mathbb{K}_{\overline{\mathcal{K}}}$, the formal context corresponding to the full context $\overline{\mathcal{K}}$. Since $Imp(\overline{\mathcal{K}}) = Imp(\mathbb{K}_{\overline{\mathcal{K}}})$, we call this also the *Duquenne-Guigues base of* $\overline{\mathcal{K}}$. Recall that the left-hand sides of the implications in this base are pseudo-intents of $\mathbb{K}_{\overline{\mathcal{K}}}$. Because the operator $(\cdot)''$ for $\mathbb{K}_{\overline{\mathcal{K}}}$ and the operator $\overline{\mathcal{K}}(\cdot)$ coincide, a subset P of M is a pseudo-intent of $\mathbb{K}_{\overline{\mathcal{K}}}$ if $P \neq \overline{\mathcal{K}}(P)$ and $\overline{\mathcal{K}}(Q) \subseteq P$ holds for all pseudo-intents $Q \subsetneq P$. We call such a set also a *pseudo-intent of* $\overline{\mathcal{K}}$.

Proposition 5.2.14. *The set $\mathcal{L}$ computed by Algorithm 4 is the Duquenne-Guigues base of $\overline{\mathcal{K}}$, and thus contains the minimum number of implications among all sets of implications that are sound and complete for $Imp(\overline{\mathcal{K}})$.*

Proof. We know that the Duquenne-Guigues base of a formal context, and thus also of the corresponding full context $\overline{\mathcal{K}}$, contains the minimum number of implications among all implication sets that are sound and complete for $Imp(\overline{\mathcal{K}})$. In Proposition 5.2.13, we have already shown that the implication set $\mathcal{L}$ produced by Algorithm 4 is sound and complete for $Imp(\overline{\mathcal{K}})$. Thus, it is enough to show that (i) the left-hand sides L of the implications in $\mathcal{L}$ are pseudo-intents of $\overline{\mathcal{K}}$, and (ii) the corresponding right-hand sides are of the form $\overline{\mathcal{K}}(L) \setminus L$.

To show (ii), consider an implication $L \to R$ in $\mathcal{L}$. By the construction of $\mathcal{L}$, there is an index i such that $R = \mathcal{K}_i(L) \setminus L$. We know that $L \to R$ is not refuted by $\overline{\mathcal{K}}$, and thus $\mathcal{K}_i(L) = \mathcal{K}_{i+1}(L) = \ldots = \mathcal{K}(L)$ by Proposition 5.2.12. Thus, it is enough to show that $\mathcal{K}(L) = \overline{\mathcal{K}}(L)$. The inclusion $\overline{\mathcal{K}}(L) \subseteq \mathcal{K}(L)$ follows from the fact that $\mathcal{K} \leq \overline{\mathcal{K}}$, and the inclusion in the other direction follows from the fact that $L \to \mathcal{K}_i(L) \setminus L$, and thus also $L \to \mathcal{K}(L)$, is not refuted by $\overline{\mathcal{K}}$ (see Proposition 5.2.4).

To show (i), first note that the implication $L \to \mathcal{K}_i(L) \setminus L$ is only added by the algorithm to the implication set if $L \neq \mathcal{K}_i(L)$. Together with what we have shown in the proof of (ii) above, this yields $L \neq \overline{\mathcal{K}}(L)$. To show that L is indeed a pseudo-intent of $\overline{\mathcal{K}}$, we assume that Q is a pseudo-intent of $\overline{\mathcal{K}}$ such that $Q \subsetneq L$. We must show that $\overline{\mathcal{K}}(Q) \subseteq L$. By Proposition 5.2.4, $Q \to \overline{\mathcal{K}}(Q)$ is not refuted by $\overline{\mathcal{K}}$. Since $\mathcal{L}$ is complete for $\mathit{Imp}(\overline{\mathcal{K}})$ by Proposition 5.2.13, the implication $Q \to \overline{\mathcal{K}}(Q)$ follows from $\mathcal{L}$, i.e., $\overline{\mathcal{K}}(Q) \subseteq \mathcal{L}(Q)$. In addition, $Q \subseteq L$ implies $\mathcal{L}(Q) \subseteq \mathcal{L}(L)$, and we know from the proof of Proposition 5.2.13 that L is $\mathcal{L}$-closed. Thus, we have $\overline{\mathcal{K}}(Q) \subseteq \mathcal{L}(Q) \subseteq \mathcal{L}(L) = L$, which completes the proof that L is a pseudo-intent of $\overline{\mathcal{K}}$. □

5.3 DL knowledge bases and partial contexts

Given a consistent DL knowledge base $(\mathcal{T}, \mathcal{A})$, any individual in $\mathcal{A}$ induces a partial object description, where the set of attributes consists of concepts. To be more precise, let M be a finite set of concept descriptions. Any individual name a occurring in $\mathcal{A}$ gives rise to the partial object description

$$\mathit{pod}_{\mathcal{T},\mathcal{A}}(a, M) := (A, S) \text{ where } \begin{array}{l} A := \{C \in M \mid \mathcal{T}, \mathcal{A} \models C(a)\} \text{ and} \\ S := \{C \in M \mid \mathcal{T}, \mathcal{A} \models \neg C(a)\}, \end{array}$$

and the whole ABox induces the partial context

$$\mathcal{K}_{\mathcal{T},\mathcal{A}}(M) := \{\mathit{pod}_{\mathcal{T},\mathcal{A}}(a, M) \mid a \text{ is an individual name occurring in } \mathcal{A}\}.$$

Note that $\mathit{pod}_{\mathcal{T},\mathcal{A}}(a, M)$ is indeed a pod since $(\mathcal{T}, \mathcal{A})$ was assumed to be consistent, and thus we cannot simultaneously have $\mathcal{T}, \mathcal{A} \models C(a)$ and $\mathcal{T}, \mathcal{A} \models \neg C(a)$.

Similarly, any element $d \in \Delta^{\mathcal{I}}$ of an interpretation $\mathcal{I}$ gives rise to the full example

$$\mathit{fod}_{\mathcal{I}}(d, M) := (\overline{A}, \overline{S}) \text{ where } \begin{array}{l} \overline{A} := \{C \in M \mid d \in C^{\mathcal{I}}\} \text{ and} \\ \overline{S} := \{C \in M \mid d \in (\neg C)^{\mathcal{I}}\}, \end{array}$$

and the whole interpretation induces the full context

$$\mathcal{K}_{\mathcal{I}}(M) := \{\mathit{fod}_{\mathcal{I}}(d, M) \mid d \in \Delta^{\mathcal{I}}\}.$$

Note that $\mathit{fod}_{\mathcal{I}}(d, M)$ is indeed a fod since every $d \in \Delta^{\mathcal{I}}$ satisfies either $d \in C^{\mathcal{I}}$ or $d \in \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} = (\neg C)^{\mathcal{I}}$.

Proposition 5.3.1. *Let $(\mathcal{T}, \mathcal{A})$ be a consistent knowledge base, M a set of concept descriptions, and $\mathcal{I}$ a model of $(\mathcal{T}, \mathcal{A})$. Then $\mathcal{K}_{\mathcal{I}}(M)$ is a realizer of $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$.*

Proof. Consider a pod $(A, S) \in \mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$, i.e., $(A, S) = pod_{\mathcal{T},\mathcal{A}}(a, M)$, where a is an individual name occurring in $\mathcal{A}$. We claim that (A, S) is realized by $(\overline{A}, \overline{S}) := fod_{\mathcal{I}}(a^{\mathcal{I}}, M) \in \mathcal{K}_{\mathcal{I}}(M)$.

Let C be an element of A, i.e., $\mathcal{T}, \mathcal{A} \models C(a)$. Since $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{A})$, this implies $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and thus $C \in \overline{A}$. This shows $A \subseteq \overline{A}$. The inclusion $S \subseteq \overline{S}$ can be shown in the same way. Now let C be an element of S, i.e., $\mathcal{T}, \mathcal{A} \models \neg C(a)$. Since $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{A})$, $a^{\mathcal{I}} \in \neg C(a)$, and thus $C \in \overline{S}$. This shows that $S \subseteq \overline{S}$. Thus we have proved that (A, S) is realized by $(\overline{A}, \overline{S})$. □

The notion of refutation of an implication is transferred from partial (full) contexts to knowledge bases (interpretations) in the obvious way.

Definition 5.3.2. The implication $L \to R$ over the attributes M is *refuted* by the knowledge base $(\mathcal{T}, \mathcal{A})$ if it is refuted by $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$, and it is *refuted* by the interpretation $\mathcal{I}$ if it is refuted by $\mathcal{K}_{\mathcal{I}}(M)$. If an implication is not refuted by $\mathcal{I}$, then we say that it *holds in* $\mathcal{I}$. The set of implications over M that hold in $\mathcal{I}$ is denoted by $Imp_M(\mathcal{I})$. In addition, we say that $L \to R$ *follows from* $\mathcal{T}$ if $\sqcap L \sqsubseteq_{\mathcal{T}} \sqcap R$, where $\sqcap L$ and $\sqcap R$ respectively stand for the conjunctions $\bigsqcap_{C \in L} C$ and $\bigsqcap_{D \in R} D$. ◇

Obviously, $L \to R$ is refuted by $(\mathcal{T}, \mathcal{A})$ iff there is an individual name a occurring in $\mathcal{A}$ such that $\mathcal{T}, \mathcal{A} \models C(a)$ for all $C \in L$ and $\mathcal{T}, \mathcal{A} \models \neg D(a)$ for some $D \in R$. Similarly, $L \to R$ is refuted by $\mathcal{I}$ iff there is an element $d \in \Delta^{\mathcal{I}}$ such that $d \in C^{\mathcal{I}}$ for all $C \in L$ and $d \notin D^{\mathcal{I}}$ for some $D \in R$. In addition, the implication $L \to R$ holds in $\mathcal{I}$ iff $(\sqcap L)^{\mathcal{I}} \subseteq (\sqcap R)^{\mathcal{I}}$.

Proposition 5.3.3. *Let $\mathcal{T}$ be a TBox and $\mathcal{I}$ be a model of $\mathcal{T}$. If the implication $L \to R$ follows from $\mathcal{T}$, then it holds in $\mathcal{I}$.*

Proof. If $L \to R$ follows from $\mathcal{T}$, then by Definition 5.3.2, $\sqcap L \sqsubseteq_{\mathcal{T}} \sqcap R$ holds. Since $\mathcal{I}$ is a model of $\mathcal{T}$, $(\sqcap L)^{\mathcal{I}} \subseteq (\sqcap R)^{\mathcal{I}}$ holds as well. This shows that $L \to R$ holds in $\mathcal{I}$. □

The operator $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)(\cdot)$ induced by the partial context $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$ is defined as in Proposition 5.2.4. Since in the following the attribute set M can be assumed to be fixed, we will write $\mathcal{K}_{\mathcal{T},\mathcal{A}}$ rather than $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$. Obviously, the result of applying this operator to a set $P \subseteq M$ can be described as follows:

$$\mathcal{K}_{\mathcal{T},\mathcal{A}}(P) = M \setminus \bigcup \{D \in M \mid \exists a.\ P \subseteq \{C \mid \mathcal{T}, \mathcal{A} \models C(a)\} \wedge \mathcal{T}, \mathcal{A} \models \neg D(a)\}$$

By Proposition 5.2.4, $\mathcal{K}_{\mathcal{T},\mathcal{A}}(P)$ is the largest subset of M such that $P \to \mathcal{K}_{\mathcal{T},\mathcal{A}}(P)$ is not refuted by $(\mathcal{T}, \mathcal{A})$.

5.3.1 Completion of DL knowledge bases

We are now ready to define what we mean by a completion of a DL knowledge base. Intuitively, the knowledge base is supposed to describe an intended model. For a fixed set M of "interesting" concepts, the knowledge base is complete if it contains all the relevant knowledge about implications between these concepts. To be more precise, if an implication holds in the intended interpretation, then it should follow from the TBox, and if it does not hold in the intended interpretation, then the ABox should contain a counterexample. Based on the notions introduced in the previous subsection, this can formally be defined as follows.

Definition 5.3.4. Let $(\mathcal{T}, \mathcal{A})$ be a DL knowledge base, M a finite set of concept descriptions, and $\mathcal{I}$ a model of $(\mathcal{T}, \mathcal{A})$. Then $(\mathcal{T}, \mathcal{A})$ is *M-complete* (or simply *complete* if M is clear from the context) *w.r.t.* $\mathcal{I}$ if the following three statements are equivalent for all implications $L \to R$ over M:

1. $L \to R$ holds in $\mathcal{I}$;
2. $L \to R$ follows from $\mathcal{T}$;
3. $L \to R$ is not refuted by $(\mathcal{T}, \mathcal{A})$.

Let $(\mathcal{T}_0, \mathcal{A}_0)$ be a DL knowledge base that also has $\mathcal{I}$ as a model. Then $(\mathcal{T}, \mathcal{A})$ is a *completion* of $(\mathcal{T}_0, \mathcal{A}_0)$ if it is complete and extends $(\mathcal{T}_0, \mathcal{A}_0)$, i.e., $\mathcal{T}_0 \subseteq \mathcal{T}$ and $\mathcal{A}_0 \subseteq \mathcal{A}$. ◇

In order to rephrase the definition of completeness, let us say that the element $d \in \Delta^{\mathcal{I}}$ of an interpretation $\mathcal{I}$ *satisfies* the subsumption statement $C \sqsubseteq D$ if $d \notin C^{\mathcal{I}}$ or $d \in D^{\mathcal{I}}$, and that $\mathcal{I}$ *satisfies* this statement if every element of $\Delta^{\mathcal{I}}$ satisfies it. In addition, let us call the individual name a a *counterexample in* $(\mathcal{T}, \mathcal{A})$ to the subsumption statement $C \sqsubseteq D$ if $\mathcal{T}, \mathcal{A} \models C(a)$ and $\mathcal{T}, \mathcal{A} \models \neg D(a)$.

Lemma 5.3.5. *The knowledge base $(\mathcal{T}, \mathcal{A})$ is complete w.r.t. its model $\mathcal{I}$ iff the following statements are equivalent for all subsets L, R of M:*

1. *$\sqcap L \sqsubseteq \sqcap R$ is satisfied by $\mathcal{I}$;*
2. *$\sqcap L \sqsubseteq_{\mathcal{T}} \sqcap R$ holds;*
3. *$(\mathcal{T}, \mathcal{A})$ does not contain a counterexample to $\sqcap L \sqsubseteq \sqcap R$.*

In the following, we use an adaptation of the attribute exploration algorithm for partial contexts presented in Section 5.2.4 in order to compute a completion of a given knowledge base $(\mathcal{T}_0, \mathcal{A}_0)$ w.r.t. a fixed model $\mathcal{I}$ of this knowledge base. It is assumed that the *expert* has enough information about this model to be able to answer questions of the form "Is $L \to R$ refuted by

$\mathcal{I}$?". If the answer is "no," then $L \to R$ is added to the implication base computed by the algorithm. In addition, the GCI $\sqcap L \sqsubseteq \sqcap R$ is added to the TBox. Since $L \to R$ is not refuted by $\mathcal{I}$, the interpretation $\mathcal{I}$ is still a model of the new TBox obtained this way. If the answer is "yes," then the expert must extend the current ABox (by adding or refining assertions) such that the extended ABox refutes $L \to R$ and $\mathcal{I}$ is still a model of this ABox. Because of Proposition 5.3.3, before actually asking the expert whether the implication $L \to R$ is refuted by $\mathcal{I}$, we can first check whether $\sqcap L \sqsubseteq \sqcap R$ already follows from the current TBox. If this is the case, then we know that $L \to R$ cannot be refuted by $\mathcal{I}$. This completion algorithm for DL knowledge bases is described in more detail in Algorithm 5.

Note that Algorithm 5 applied to $\mathcal{T}_0, \mathcal{A}_0, M$ with the underlying model $\mathcal{I}$ of $(\mathcal{T}_0, \mathcal{A}_0)$ behaves identical to Algorithm 4 applied to the partial context $\mathcal{K}_{\mathcal{T}_0,\mathcal{A}_0}(M)$ with the underlying full context $\mathcal{K}_{\mathcal{I}}(M)$ as realizer. This is an immediate consequence of the following facts:

1. for all $i \geq 0$, the underlying interpretation $\mathcal{I}$ is a model of $(\mathcal{T}_i, \mathcal{A}_i)$;
2. if the test $\sqcap P_i \sqsubseteq_{\mathcal{T}_i} \sqcap \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ is successful, then the implication $P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ holds in $\mathcal{I}$, and thus the expert would have answered "no" to this implication question;
3. if $\mathcal{T}'$ is a TBox such that $\mathcal{T}_i \subseteq \mathcal{T}'$ and $\mathcal{I}$ is a model of $\mathcal{T}'$, then $\mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(M) \leq \mathcal{K}_{\mathcal{T}',\mathcal{A}_i}(M) \leq \mathcal{K}_{\mathcal{I}}(M)$;
4. if $\mathcal{A}'$ is an ABox such that $\mathcal{A}_i \subseteq \mathcal{A}'$, $\mathcal{I}$ is a model of $\mathcal{A}'$, and $P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ is refuted by $\mathcal{A}'$, then $\mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(M) \leq \mathcal{K}_{\mathcal{T}_i,\mathcal{A}'}(M) \leq \mathcal{K}_{\mathcal{I}}(M)$ and $P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ is refuted by $\mathcal{K}_{\mathcal{T}_i,\mathcal{A}'}(M)$.

Thus, Proposition 5.2.13 immediately implies the following proposition.

Proposition 5.3.6. *Let $(\mathcal{T}_0, \mathcal{A}_0)$ be a knowledge base, M a finite set of concept descriptions, and $\mathcal{I}$ a model of $(\mathcal{T}_0, \mathcal{A}_0)$. Then Algorithm 5 terminates, and upon termination outputs a knowledge base $(\mathcal{T}, \mathcal{A})$ and a set of implications $\mathcal{L}$ such that*

- *$\mathcal{L}$ is sound and complete for $Imp_M(\mathcal{I})$, and*
- *$(\mathcal{T}, \mathcal{A})$ refutes every implication that is refuted by $\mathcal{I}$.*

It remains to show that Algorithm 5 really computes a completion of the input knowledge base.

Theorem 5.3.7. *Let $(\mathcal{T}_0, \mathcal{A}_0)$ be a knowledge base, M a finite set of concept descriptions, and $\mathcal{I}$ a model of $(\mathcal{T}_0, \mathcal{A}_0)$, and let $(\mathcal{T}, \mathcal{A})$ be the knowledge base computed by Algorithm 5. Then $(\mathcal{T}, \mathcal{A})$ is a completion of $(\mathcal{T}_0, \mathcal{A}_0)$.*

Algorithm 5 Completion of DL knowledge bases

Input: $\mathcal{T}_0$, $\mathcal{A}_0$, M {$(\mathcal{T}_0, \mathcal{A}_0)$ has the underlying interpretation $\mathcal{I}$ as model}
$i := 0$
$\mathcal{L}_0 := \emptyset$ {initial empty set of implications}
$P_0 := \emptyset$ {lectically smallest $\mathcal{L}_0$-closed subset of M}
while $P_i \neq M$ **do**
  Compute $\mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$
  if $P_i \neq \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ **then** {check if the implication follows from $\mathcal{T}_i$}
    if $\sqcap P_i \sqsubseteq_{\mathcal{T}_i} \sqcap \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ **then**
      $\mathcal{A}_{i+1} := \mathcal{A}_i$
      $\mathcal{L}_{i+1} := \mathcal{L}_i \cup \{P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i) \setminus P_i\}$
      $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
    else
      Ask the expert if $P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ is refuted by $\mathcal{I}$.
      if no **then** {$\sqcap P_i \sqsubseteq \sqcap \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ is satisfied in $\mathcal{I}$}
        $\mathcal{A}_{i+1} := \mathcal{A}_i$
        $\mathcal{L}_{i+1} := \mathcal{L}_i \cup \{P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i) \setminus P_i\}$
        $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
        $\mathcal{T}_{i+1} := \mathcal{T}_i \cup \{\sqcap P_i \sqsubseteq \sqcap(\mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i) \setminus P_i)\}$ {extend the TBox}
      else
        Get an ABox $\mathcal{A}'$ from the expert such that $\mathcal{A}_i \subseteq \mathcal{A}'$, $\mathcal{I}$ is a model of $\mathcal{A}'$, and $P_i \to \mathcal{K}_{\mathcal{T}_i,\mathcal{A}_i}(P_i)$ is refuted by $\mathcal{A}'$
        $\mathcal{A}_{i+1} := \mathcal{A}'$ {extend the ABox}
        $\mathcal{T}_{i+1} = \mathcal{T}_i$
        $P_{i+1} := P_i$
        $\mathcal{L}_{i+1} := \mathcal{L}_i$
      end if
    end if
  else
    $\mathcal{A}_{i+1} := \mathcal{A}_i$
    $\mathcal{T}_{i+1} := \mathcal{T}_i$
    $\mathcal{L}_{i+1} := \mathcal{L}_i$
    $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ for the max. j that satisfies $P <_j \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$
  end if
  $i := i + 1$
end while

Proof. Obviously, $(\mathcal{T}, \mathcal{A})$ extends $(\mathcal{T}_0, \mathcal{A}_0)$ and has $\mathcal{I}$ as a model. To prove that $(\mathcal{T}, \mathcal{A})$ is complete, we must show that the following three statements

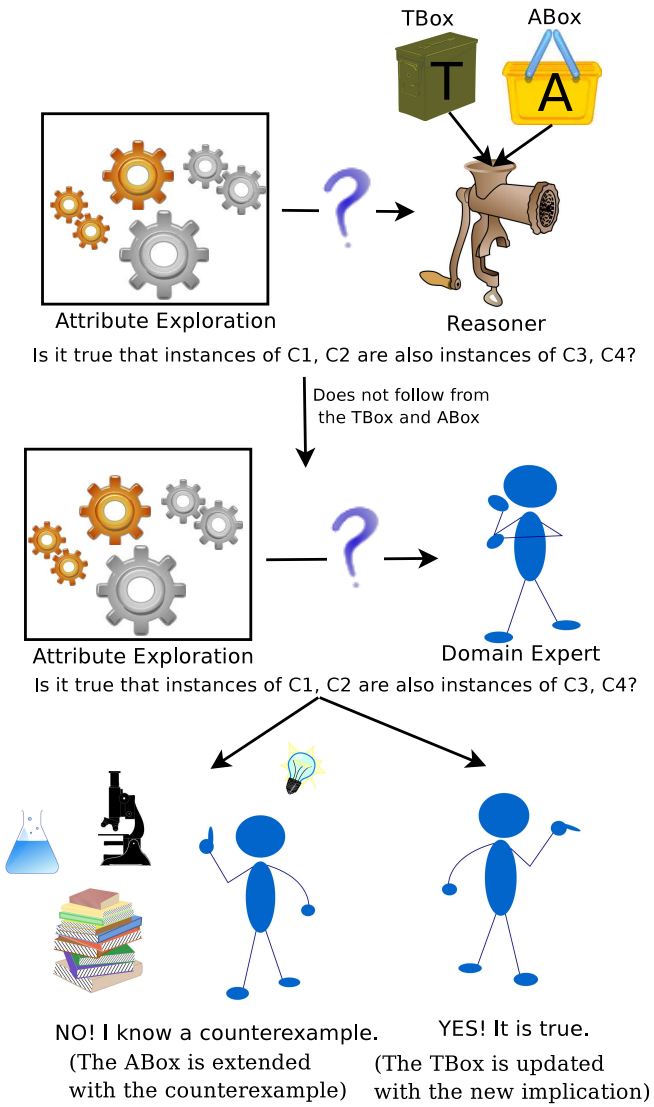

Figure 5.1: Flow of information in Algorithm 5

are equivalent for all implications $L \to R$ over M:

1. $L \to R$ holds in $\mathcal{I}$;
2. $L \to R$ follows from $\mathcal{T}$;
3. $L \to R$ is not refuted by $(\mathcal{T}, \mathcal{A})$.

"2 $\to$ 1" is an immediate consequence of the fact that $\mathcal{I}$ is a model of $\mathcal{T}$, and "1 $\to$ 2" follows from the facts that $\mathcal{L}$ is complete for $Imp_M(\mathcal{I})$ and $\mathcal{T}$ contains the GCIs $\sqcap L' \sqsubseteq \sqcap R'$ for all implications $L' \to R'$ in $\mathcal{L}$.

$\mathcal{A}_{countries}$	Asian	EU	European	G8	Mediterranean
Syria	+	-	-	-	+
Turkey	+	-	+	-	+
France	-	+	+	+	+
Germany	-	+	+	+	-
Switzerland	-	-	+	-	-
USA	-	-	-	+	-

Table 5.1: The partial context before completion

"$1 \rightarrow 3$" is an immediate consequence of the fact that $\mathcal{I}$ is a model of $\mathcal{A}$, and "$3 \rightarrow 1$" of the fact that $(\mathcal{T}, \mathcal{A})$ refutes every implication that is refuted by $\mathcal{I}$. □

Example 5.3.8. Let us demonstrate the execution of Algorithm 5 on a small knowledge base. Let our TBox $\mathcal{T}_{countries}$ contain the following concept definitions:

$$\begin{aligned}
\mathsf{AsianCountry} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasTerritoryIn}.\{Asia\} \\
\mathsf{EUmember} &\equiv \mathsf{Country} \sqcap \exists \mathsf{memberOf}.\{EU\} \\
\mathsf{EuropeanCountry} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasTerritoryIn}.\{Europe\} \\
\mathsf{G8member} &\equiv \mathsf{Country} \sqcap \exists \mathsf{memberOf}.\{G8\} \\
\mathsf{IslandCountry} &\equiv \mathsf{Country} \sqcap \neg\exists \mathsf{hasTerritoryIn}.\mathsf{Continent} \\
\mathsf{MediterrenaenCountry} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasBorderTo}.\{MediterrenaenSea\}
\end{aligned}$$

Moreover, let our ABox $\mathcal{A}_{countries}$ contain the individuals *Syria, Turkey, France, Germany, Switzerland, USA* and assume we are interested in the subsumption relationships between the concept names AsianCountry, EUmember, EuropeanCountry, G8member and MediterreneanCountry. Table 5.1 shows the partial context induced by $\mathcal{A}_{countries}$, and Table 5.2 shows the questions asked by the completion algorithm and the answers given to these questions. In order to save space, the names of the concepts are shortened in both tables. The questions with positive answers result in extension of the TBox with the following GCIs:

$$\begin{aligned}
\mathsf{G8member} \sqcap \mathsf{MediterraneanCountry} &\sqsubseteq \mathsf{EUmember} \sqcap \mathsf{EuropeanCountry} \\
\mathsf{EUmember} \sqcap \mathsf{G8member} &\sqsubseteq \mathsf{EuropeanCountry} \\
\mathsf{AsianCountry} \sqcap \mathsf{EUmember} &\sqsubseteq \mathsf{MediterraneanCountry} \\
\mathsf{AsianCountry} \sqcap \mathsf{EUmember} \sqcap & \\
\mathsf{EuropeanCountry} \sqcap \mathsf{MediterraneanCountry} &\sqsubseteq \mathsf{G8member}
\end{aligned}$$

Moreover, the questions with negative answers result in extension of the ABox with the individuals *Russia, Cyprus, Spain* and *Japan*. The partial context induced by the resulting ABox $\mathcal{A}'_{countries}$ is shown in Table 5.3.

Question	Answer	Counterex.
{G8, Mediterranean} → {EU, European}?	yes	-
{European, G8} → {EU}?	no	Russia
{EU} → {European, G8}?	no	Cyprus
{EU, G8} → {European}?	yes	-
{EU, European} → {G8}?	no	Spain
{Asian, G8} → {European}?	no	Japan
{Asian, EU} → {Mediterranean}?	yes	-
{Asian, EU, European, Mediterranean} → {G8}?	yes	-

Table 5.2: Execution of Algorithm 5 on $(\mathcal{T}_{countries}, \mathcal{A}_{countries})$

The resulting knowledge base $(\mathcal{T}'_{countries}, \mathcal{A}'_{countries})$ is complete w.r.t. the initially selected concept names.

$\mathcal{A}'_{countries}$	Asian	EU	European	G8	Mediterranean
Syria	+	-	-	-	+
Turkey	+	-	+	-	+
France	-	+	+	+	+
Germany	-	+	+	+	-
Switzerland	-	-	+	-	-
USA	-	-	-	+	-
Russia	+	-	+	+	-
Cyprus	+	+	-	-	+
Spain	-	+	+	-	+
Japan	+	-	-	+	-

Table 5.3: The partial context after completion

◇

5.4 Implementation

Based on the results presented in the previous two sections, we have implemented a first experimental version of a DL knowledge base completion tool called InstExp [2], which stands for INSTance EXPlorer. It implements Algorithm 5 given in the previous section. InstExp is written in the Java programming language as an extension to version v2.3 beta 3 of the Swoop ontology editor [KPS+06]. It communicates with the reasoner over the OWL API [BVL03].

[2] available under `http://lat.inf.tu-dresden.de/~sertkaya/InstExp/`

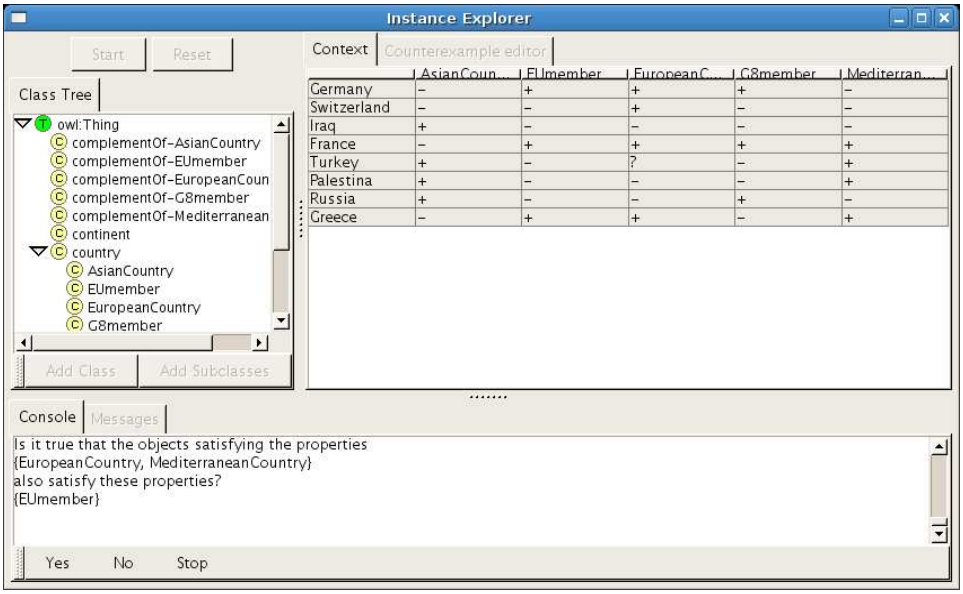

Figure 5.2: InstExp window during completion

Usage

A DL knowledge base completion process using InstExp can briefly be sketched as follows: after loading a knowledge base into Swoop and classifying it, the user can start InstExp from the Swoop menu. At this point InstExp displays the concept hierarchy of the loaded knowledge base, and asks the user to select the "interesting" concepts that should be involved in the completion process. As soon as the user finishes selecting these concepts, InstExp displays the partial context containing the individuals in the ABox that are instances of some of these concepts, and the completion process starts with the first question. Figure 5.2 shows a snapshot of the InstExp window during the completion process. The user can confirm or reject the questions by clicking the corresponding buttons. If she rejects a question, InstExp displays a counterexample editor, which contains the "potential counterexamples" to the current question, i.e., individuals in the ABox that can be modified to act as a counterexample to this question. The user can either modify one of these existing individuals and turn it into a counterexample, or introduce a new individual into the ABox. During counterexample preparation, InstExp tries to guide the user as follows: If she makes the description of an individual inconsistent, InstExp gives a warning and does not allows her to provide this as the description of a counterexample. Once she has produced a description that is sufficient to act as a counterexample, InstExp notifies the user about this, and allows this description to be added to the ABox. Figure 5.3 shows a snapshot of the

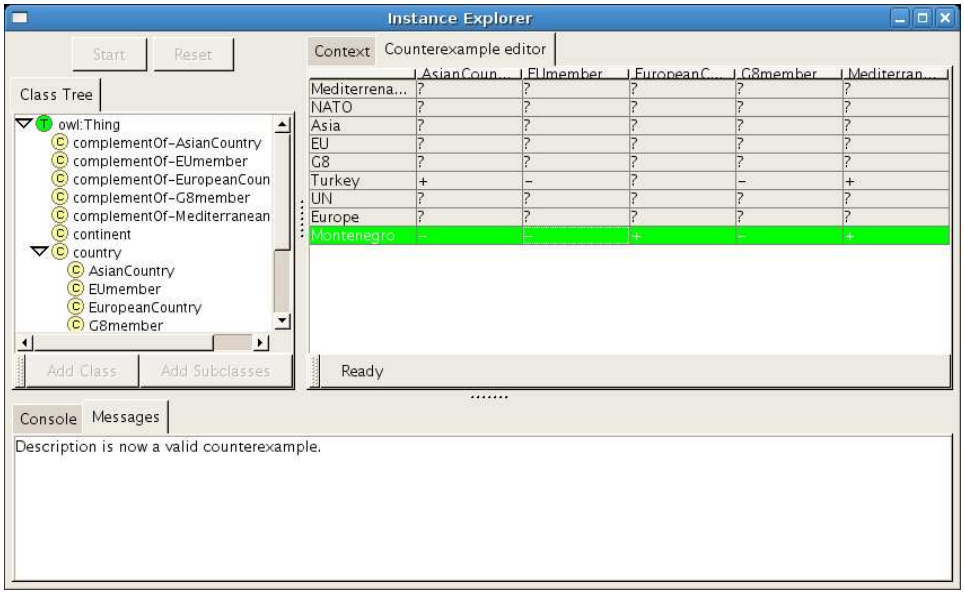

Figure 5.3: InstExp window during counterexample preparation

InstExp window during counterexample preparation.

Experiments

In order to evaluate the performance and usability of InstExp we have made a series of experiments. We have observed that the performance of InstExp heavily depends on the knowledge base to be completed and the efficiency of the DL reasoner used. As already mentioned, whenever a question is accepted, a new GCI is added to the TBox. This requires the knowledge base to be reclassified. Depending on the size and complexity of the knowledge base, and efficiency of the underlying DL reasoner, this can take a long time for real life knowledge bases, which means that the user may have to wait several minutes between two consecutive questions. To some extent this problem can be overcome by using a DL reasoner that can perform incremental reasoning, i.e., that can efficiently handle the added GCI and reclassify the knowledge base without starting from scratch. Pellet can do incremental reasoning to some extent.

Another point we have observed is that, during completion, unsurprisingly the expert sometimes makes errors when answering questions. In the simplest case, the error makes the knowledge base inconsistent, which can easily be detected by DL reasoning and the expert can be notified about it. However, in this case an explanation for the reason of inconsistency is often needed to understand and fix the error. The situation gets more complicated if the error does not immediately lead to an inconsistency, but the expert

realizes only in a later step that she has done something wrong in one of the previous steps. In this case, the tool should be able to help the expert to detect which one of the previous answers leads to the error. Once the source of the error is found, the next task is to correct it without producing extra work for the expert. More precisely, the naive idea of going back to the step where the error was made, and forgetting all answers given after this step will result in asking some of the questions again. The tool should avoid this. A more sophisticated approach to minimize the effort for fixing the error is rather involved and it requires Algorithm 5 to be modified accordingly.

We have also observed that, in some cases the expert wants to skip a question, and proceed with another one. On the FCA theory side, this is not an easy task. It needs the particular lexical order used in the algorithm to be modified. Doing it in a naive way might result in the loss of soundness or completeness of the algorithm. This issue, together with other usability issues mentioned above, are going to be considered in the future. At the moment, unfortunately we do not yet have experimental results from real world applications. We have just finished implementing the first prototype of our tool, and made the above observations in our experiments with small knowledge bases. After solving the usability issues mentioned above, we are going to evaluate our tool on the OWL ontology for human protein phosphatases.

Chapter 6

On the Generators of Closed Sets

In the present chapter we investigate the problem of finding the generators of a closed set. More precisely, we are interested in finding "small" sets that generate a given closed set under a given closure operator. In particular we consider the two closure operators introduced in Chapter 3 namely, the closure operator induced by a set of implications (Section 3.3), and the closure operator induced by a formal context (Section 3.2). By saying small sets, we mean two notions of minimality: sets minimal w.r.t set inclusion, and sets minimal w.r.t cardinality. Solving the first problem, which is finding the generators of a set closed under a set of implications, can help the expert during attribute exploration by making the implication questions contain less attributes. The second problem, which is the problem of finding the generators of a concept intent, has been considered in the literature due to its role in lattice construction and merge algorithms. Here we analyze the computational complexity of some decision and counting problems related to these two problems. We start with Section 6.1, where we give an introduction to the complexity of counting. In Section 6.2 we formally define the problems mentioned above. In Section 6.3 we focus on the generators of sets closed under implications, and show that the problems of counting minimal and minimum cardinality generators are both intractable. In Section 6.4 we focus on the generators of concept intents, and show that the same results also apply in this setting.

6.1 Complexity of counting

Complexity theory typically deals with decision problems. These are problems that ask whether a solution exists. For instance, the decision problem SAT is the problem of deciding whether a Boolean expression given in conjunctive normal form is satisfiable. In computational complexity theory there is an extensive literature on the complexity of decision problems, two

of which are the well known monographs [GJ90, Pap94].

Apart from decision problems, there is another important, natural, and fundamentally different type of problems that ask "*how many different solutions exist*" for a given problem instance. These are called *counting problems*. For instance, the counting version of SAT called #SAT is the following problem: given a Boolean expression in conjunctive normal form, compute the number of truth assignments that satisfy this expression. Obviously, a counting problem is at least as hard as its underlying decision problem. For instance, if we can solve #SAT, then we can also solve SAT. Because an expression is satisfiable if and only if the number of truth assignments that satisfy it is non-zero.

6.1.1 The counting complexity class #P

A counting problem is usually defined by a *witness function* that, for every input, returns a set of witnesses. Formally, a witness function and a counting problem are defined as follows:

Definition 6.1.1 (Counting problem). Let $w : \Sigma^* \to \mathcal{P}^{<\omega}(\Gamma^*)$, where Σ and Γ are two alphabets, and $\mathcal{P}^{<\omega}(\Gamma^*)$ is the set of all finite subsets of Γ^*. We call w a *witness function*. Every witness function gives rise to the following *counting problem*: given a string $x \in \Sigma^*$, find the cardinality of the witness set $w(x)$, i.e., $|w(x)|$. ◇

The complexity of counting problems was first investigated by Valiant [Val79a]. For systematically studying and classifying counting problems, he introduced the counting complexity class #P, which is the class of functions that count the number of accepting paths of nondeterministic polynomial-time Turing machines. Typical members of this class are the problems of counting the number of solutions of NP-complete problems. Among them, the prototypical problem in #P is #SAT, which is the following problem:

Problem 6.1.2. *(#SAT)*
INSTANCE: A Boolean formula φ in conjunctive normal form.
OUTPUT: The number of all distinct truth assignments that satisfy φ.

In [Val79a], Valiant showed that #SAT is #P-complete. Moreover, in [Val79b] he demonstrated that #P contains other natural complete problems, for which there was no previous indication of intractability. Among these problems are #PERFECT MATCHINGS[1], which is the problem of counting the perfect matchings of a bipartite graph, #MINIMAL VERTEX COVER, which is the problem of counting the minimal vertex covers of a graph, and #MONOTONE 2-SAT, which is the problem of counting the binary vectors that satisfy a monotone 2-CNF formula.

[1]This problem is also known as #PERMANENT since it is equivalent to computing the permanent of the adjacency matrix of a bipartite graph

Valiant's most surprising discovery, however, was that there can be a dramatic gap in computational complexity between a counting problem and its underlying decision problem. To be more precise, he showed that there are #P-complete problems whose underlying decision problem can be solved in polynomial time. The first problem that was shown to have this *easy to decide, but hard to count* property was #PERFECT MATCHINGS. It is well known that given a bipartite graph, whether it has a perfect matching can be decided in polynomial time. However, as shown in [Val79b], counting the perfect matchings of a given bipartite graph is #P-complete, i.e., intractable.

6.1.2 Beyond #P

In addition to introducing #P, in [Val79a] Valiant gave the following definition for counting complexity classes beyond #P.

Definition 6.1.3 (Classes beyond #p). For any complexity class C, define $\#\mathrm{C} = \cup_{A \in \mathrm{C}} (\#\mathrm{P})^A$, where by $(\#\mathrm{P})^A$ we mean the functions counting the accepting paths of nondeterministic polynomial-time Turing machines having A as their oracle. ◇

According to this definition, the first class that comes after #P is the class #NP, which is the class of functions that count the number of accepting paths of $\mathrm{NP}^{\mathrm{NP}}$ machines, i.e, nondeterministic polynomial-time Turing machines with NP oracles. Definition 6.1.3 has the following interesting property: since it does not matter whether an oracle or its complement is called, according to this definition $\#\mathrm{C} = \#\mathrm{coC}$ holds for every complexity class C. More precisely, $\#\Sigma_k\mathrm{P} = \#\Pi_k\mathrm{P}$ for every $k \geq 1$, where $\Sigma_k\mathrm{P}$ is the kth level of the polynomial hierarchy, and $\Pi_k\mathrm{P} = co\Sigma_k\mathrm{P}$. In particular, for the first level of the polynomial hierarchy ($k = 1$), $\#\mathrm{NP} = \#\mathrm{coNP}$.

In order to make a finer classification of counting problems, in [HV95] Hemaspaandra and Vollmer have introduced a predicate-based approach for defining higher complexity classes. In this approach, the counting complexity classes are named as $\#\cdot\mathrm{C}$ in order to avoid confusion with Valiant's notation. Hemaspaandra and Vollmer define the higher counting complexity classes as follows:

Definition 6.1.4. For any complexity class C, define $\#\cdot\mathrm{C}$ to be the class of functions f such that, for some C-computable two-argument predicate R and some polynomial p, for every string x it holds that:

$$f(x) = |\{y \mid p(|x|) = |y| \text{ and } R(x, y)\}|$$

◇

Informally, $\#\cdot\mathrm{C}$ is the class of functions counting the number of appropriate-length second arguments that cause some C predicate to be true for the given first argument. It captures the class of counting problems whose witness function w satisfies the following conditions:

1. There is a polynomial $p(n)$ such that for every x and every $y \in w(x)$ we have $|y| \leq p(|x|)$,

2. The decision problem "*given x and y, is $y \in w(x)$?*" is in C.

It is easy to verify that the counting complexity class $\#\cdot$P in this approach is equivalent to the counting complexity class #P in Valiant's approach, i.e, $\#\cdot\mathrm{P} = \#\mathrm{P}$. However, in general $\#\mathrm{C} = \# \cdot \mathrm{C}$ need not hold for every complexity class C. The precise relationship between the complexity classes of these two different approaches was shown to be as follows by Toda in [Tod91]:

$$\# \cdot \Sigma_k\mathrm{P} \subseteq \#\Sigma_k\mathrm{P} = \# \cdot \mathrm{P}^{\Sigma_k\mathrm{P}} = \# \cdot \Pi_k\mathrm{P}$$

for every $k \geq 1$. In particular, for the first level of the polynomial hierarchy ($k = 1$), $\# \cdot \mathrm{NP} \subseteq \#\mathrm{NP} = \# \cdot \mathrm{P}^{\mathrm{NP}} = \# \cdot co\mathrm{NP}$. This result shows that the predicate-based approach by Hemaspaandra and Vollmer makes a finer classification of counting problems.

In general, what makes a complexity class interesting is whether there exist natural, complete problems for this class. We have already mentioned that, in this sense, #P is an interesting class since it contains several natural, complete problems like #PERFECT MATCHINGS, #MINIMAL VERTEX COVER, etc. For higher counting complexity classes, the prototypical problems are counting the satisfying truth assignments of quantified Boolean formulae with a bounded number of quantifier alterations. For a positive integer k, the counting problem $\#\Pi_k$SAT is defined as follows:

Problem 6.1.5. *($\#\Pi_k$SAT)*
INSTANCE: A formula

$$\varphi(y_1, \ldots, y_n) = \forall x_1 \exists x_2 \ldots Q_k x_k \psi(x_1, \ldots, x_k, y_1, \ldots, y_n),$$

where ψ is a Boolean formula, each x_i is a tuple of variables, and each y_j is a variable.

OUTPUT: The number of distinct truth assignments to the variables $y_1, \ldots, y_n$ that satisfy φ.

The counting problem $\#\Sigma_k$SAT is defined similarly using a formula of the form $\exists x_1 \forall x_2 \ldots Q_k x_k \psi(x_1, \ldots, x_k, y_1, \ldots, y_n)$, where ψ is a Boolean formula, each x_i is a tuple of variables, and each y_j is a variable. It is known that the problem $\#\Pi_k$SAT is $\#\cdot\Pi_k$P-complete, and the problem $\#\Sigma_k$SAT is $\#\cdot\Sigma_k$P-complete [DHK05].

6.1.3 Subtractive reductions

Completeness of the problems in #P is usually proved by using *parsimonious reductions*, which are polynomial-time reductions that preserve the number of solutions by establishing a bijection between the solution sets of

the problems. Parsimonious reductions have shortcomings when one wants to use them for showing completeness of problems in counting complexity classes higher than #P. In addition to preserving the number of solutions, they also preserve the complexity of the underlying decision problem; thus they cannot be used to discover the existence of problems that are complete for higher counting complexity classes and have the "easy to decide, hard to count" property mentioned above. Due to this fact, in [DHK05] Durand et al. have introduced a new kind of reduction called *subtractive reduction*, such that #P and higher counting complexity classes are closed under it, and it does not establish a direct bijection between the solution sets of the problems considered. Hence, they can be used to reduce counting problems whose underlying decision problems are of different complexity.

A subtractive reduction from a counting problem to another counting problem first overcounts the number of solutions, and then carefully subtracts any surplus. It is formally defined as follows:

Definition 6.1.6 (Subtractive reduction). Let #A and #B be two counting problems determined by the binary relations A and B between strings from Σ and Γ. We say that #A reduces to #B via a *strong subtractive reduction* if there exist two polynomial-time computable functions f and g such that for every string $x \in \Sigma^*$ the following conditions hold:

- $B(f(x)) \subseteq B(g(x))$
- $|A(x)| = |B(g(x))| - |B(f(x))|$.

$\diamond$

Parsimonious reductions are a special case of subtractive reductions, where $B(f(x)) = \emptyset$. In [DHK05] it was pointed out that subtractive reductions are perfect tools to study higher counting complexity classes #·coNP and above. There it was shown that these higher counting complexity classes are closed under subtractive reductions, and that subtractive reductions can be used to discover natural problems that are complete for higher counting complexity classes, and problems that have the "easy to decide, hard to count" property. The first problem shown to have this property is the problem called #CIRCUMSCRIPTION, which is the problem of counting the minimal models of a Boolean formula. In [DHK05] it was shown that #CIRCUMSCRIPTION is #·coNP-complete via a subtractive reduction from #Π_1SAT. In [HP07] a detailed analysis of counting problems of propositional abduction was made, and some of these problems were also shown to be #·coNP-complete.

6.2 Small generators of closed sets

In this section, we investigate the problem of finding generators of a closed set. More precisely, given a closure operator φ on a set G and a set $A \subseteq$

G such that $\varphi(A) = A$, we are interested in finding "small" sets $B \subseteq A$ such that $\varphi(B) = A$. In particular, we consider the following two closure operators introduced in Chapter 3: 1) the closure operator $\mathcal{L}(\cdot)$ induced by a set of implications (from Section 3.3), and 2) the closure operator $(\cdot)''$ induced by a formal context (from Section 3.2). By saying small sets we mean the following two notions of minimality: 1) sets minimal w.r.t. set inclusion, 2) sets minimal w.r.t. cardinality. Based on these two notions of minimality, we investigate the problem for the two closure operators above.

Solving the first problem, which is the problem of finding small generators of a set closed under implications, can help the expert during attribute exploration by making the implication questions contain less attributes. In Section 3.3, we have mentioned that the Duquenne-Guigues Base contains the minimum number of implications, and thus the attribute exploration algorithm asks the minimum number of questions to the expert. This implies that none of the implications in the Duquenne-Guigues Base is redundant. However, it is possible that an implication in the Duquenne-Guigues Base can be shortened by removing redundant attributes in its premise and conclusion. A very similar problem has been considered in the field of relational databases [Mai83] a long time ago. The problem considered there is known as finding the keys of a relation. In [LO78] it has been shown that checking whether a relation has a key of size less than a specified integer is NP-complete. Here we show the following results: given a set of implications $\mathcal{L}$ on M, a set $A \subseteq M$ such that $\mathcal{L}(A) = A$, and a positive integer $m \leq |M|$

- the NP-completeness result in [LO78] applies to the problem of deciding whether A has a generator of size less than or equal to m (Proposition 6.3.5)
- the problem of counting minimal generators of A is #P-complete (Theorem 6.3.8)
- the problem of counting minimum cardinality generators of A is $\#\cdot$coNP-complete (Theorem 6.3.10)

The second problem, which is the problem of finding small generators of a set closed under $(\cdot)''$, is known as finding the minimal generators of a concept intent. Different aspects of it have been considered in the literature [NVRG05, FVG05]. Minimal generators of a concept intent play an important role in incremental lattice construction, and lattice merge algorithms. In [NVRG05], the behaviour of minimal generators upon increases in the underlying context's attribute set has been investigated, and a method for computing the family of minimal generators has been presented. Here, we show the following results: Given a formal context $\mathbb{K} = (G, M, I)$, a set $A \subseteq M$ such that $A'' = A$, and a positive integer $k \leq |M|$

- the problem of checking if A has a generator of size less than or equal to k is NP-complete (Theorem 6.4.2)

- the problem of counting minimal generators of A is #P-complete (Theorem 6.4.4)

6.3 Small generators of implication closed sets

Before we start, for the sake of clarity, let us recall two different notions of minimality for sets. We say that a set S is *minimal* w.r.t. property P if S satisfies P, and no $S' \subsetneq S$ satisfies P. We say that S is of *minimum cardinality* w.r.t property P if S satisfies P and no other set S' such that $|S'| < |S|$ satisfies P. From the definition, it is not difficult to see that being of minimum cardinality implies being minimal, but the other direction of the implication does not necessarily hold. Based on these notions of minimality, in this section we are going to look for answers to the following two questions: during attribute exploration whenever an implication question arises, is it possible to shorten the implication such that the premise and conclusion of the implication are

- minimal,
- of minimum cardinality?

Let us formalize the problem in a more general setting as follows:

Problem 6.3.1. *Given a set of implications $\mathcal{L}$ on M, and an $\mathcal{L}$-closed subset P of M,*

- *find a minimal subset Q of P such that Q generates P, i.e., $\mathcal{L}(Q) = P$*
- *find a minimum cardinality subset Q of P such that Q generates P.*

In the following, we will say that Q is a *minimal generator* of P under $\mathcal{L}$ if it is an answer of the first question, and *minimum cardinality generator* of P under $\mathcal{L}$ if it is an answer of the second question. The first question does not seem to be difficult. In order to obtain a minimal generator of P, we can start with P, iterate over all elements of P, and remove an element if the remaining set still generates P. Algorithm 6 states the idea formally. It determines a minimal generator of a given P under a given $\mathcal{L}$.

Algorithm 6 terminates since P is finite. Upon termination, Q is a minimal generator of P since it does not contain any redundant attribute. At step 6, for checking whether $Q \setminus \{m\}$ generates P, we can use the implicational closure algorithm mentioned in Section 3.3. This algorithm runs in $O(|\mathcal{L}||M|)$. Algorithm 6 makes at most $|M|$ iterations, as a result, it runs in $O(|\mathcal{L}||M|^2)$.

The second question in Problem 6.3.1 seems to be more challenging. Obviously, removing redundant attributes as above might not result in a

Algorithm 6 Minimal generator

```
1: Initialization
2: L                          {Given set of implications on the attribute set M}
3: P                          {Given P ⊆ M such that L(P) = P}
4: Q := P
5: for all m ∈ P do
6:    if L(Q \ {m}) = P then {if Q \ {m} generates P}
7:       Q := Q \ {m}
8:    end if
9: end for
10: return Q
```

minimum cardinality generator. There might be a generator Q' incomparable with the minimal generator Q we would obtain as above, and the cardinality of Q' might be less than the cardinality of Q. Algorithm 6 can not find such generators. A very similar problem has been considered in the field of relational databases a long time ago [LO78] under the name KEY OF CARDINALITY M problem[2]. In the following, we are going to look at this problem and try to find an answer to our second question based on it. Before we start, let us first recall some basic notions of relational databases.

6.3.1 Relation to relational databases

In relational databases [Mai83], functional dependencies are a way of expressing constraints on data. Informally, a functional dependency occurs when the values of a tuple on one set of attributes uniquely determine the values on another set of attributes. Formally, given a relation r and a set of attribute names R, a *functional dependency* (FD) is a pair of sets $X, Y \subseteq R$ written as $X \to Y$. The relation r *satisfies* the FD $X \to Y$ if the tuples with equal X-values also have equal Y-values. In this case, we say that the set of attributes X *functionally determine* the set of attributes Y.

The number of FDs that a relation satisfies is finite, since there is only a finite number of subsets of R. Thus it is always possible to find all FDs that a relation satisfies, by trying all pairs of subsets of R. However, this naïve approach is very inefficient. Instead, knowing a set of FDs satisfied by a relation enables us to infer the other FDs satisfied by this relation. We say that a set of FDs F *implies* $X \to Y$, if every relation that satisfies all FDs in F also satisfies $X \to Y$, and we write this as $F \models X \to Y$. This semantic notion of implication has also its syntactic counterpart which is a sound and complete set of inference axioms known as the *Armstrong's axioms* [Arm74]. It is the following set of axioms:

[2]This problem is called MINIMUM CARDINALITY KEY problem in [GJ90]

Definition 6.3.2 (Armstrong's axioms). Let r be a relation on the attribute set R, and X, Y, Z, W be subsets of R, and F be a set of FDs that are satisfied by r. Then,

- **Reflexivity**: $X \rightarrow X \in F$
- **Augmentation**: $X \rightarrow Y \in F$ implies $X \cup Z \rightarrow Y \in F$
- **Pseudotransitivity**: $X \rightarrow Y \in F$ and $Y \cup Z \rightarrow W \in F$ imply $X \cup Z \rightarrow W \in F$

$\diamond$

One other important concept in relational databases is the notion of a key. A subset K of the attribute set R is called a *key* of the relation r if K functionally determines R. It is called a minimal key, if no proper subset of it is a key. Alternatively, given a set of functional dependencies F that are satisfied by r, K is called a key of the relational system $< R, F >$ if $K \rightarrow R$ can be inferred from F by using Armstrong's axioms. In practical applications, it is important to find "small" keys of a given relation. In [LO78] an algorithm that determines all minimal keys of a given relation was given. There, it was also analyzed how difficult it is to check whether a given relation has a key of cardinality less than or equal to a given integer. This problem, as mentioned above, is known as the KEY OF CARDINALITY M problem. Let us call it KCM for short. It is defined as follows:

Problem 6.3.3. *(*KEY OF CARDINALITY M*)*
INSTANCE: A set R of attribute names, a set F of FDs, and a positive integer $m \leq |R|$.
QUESTION: Is there a key of cardinality m or less for the relational system $< R, F >$?

In [LO78], it was shown that KCM is NP-complete. The problem of finding a minimum cardinality generator of a closed set, mentioned as the second question in Problem 6.3.1, seems to be very similar to KCM. In order to exploit this similarity and analyze our problem, we formulate the decision version of it as follows, and call it the GENERATOR OF CARDINALITY M problem, or GCM for short.

Problem 6.3.4. *(*GENERATOR OF CARDINALITY M *of an implication-closed set)*
INSTANCE: A set M of attribute names, a set $\mathcal{L}$ of implications on M, an $\mathcal{L}$-closed subset P of M, and a positive integer $m \leq |M|$.
QUESTION: Is there a subset Q of P of cardinality m or less that generates P under $\mathcal{L}$, i.e., is there a $Q \subseteq P$ such that $\mathcal{L}(Q) = P$ and $|Q| \leq m$?

It is not difficult to see that KCM is a special case of GCM. Without loss of generality, we can think of FDs as implications, and the notion of computing closure under FDs as implicational closure. In GCM, we are interested in finding a small generator of a given closed set. Similarly, in KCM, one is interested in finding a small generator of the whole attribute set, which is the largest closed set under the given set of functional dependencies. Thus if we can solve GCM for any given closed set, we can also solve KCM by choosing the whole attribute set as the closed set given as input. Thus, GCM is NP-hard. For the sake of clarity, let us write it down formally.

Proposition 6.3.5. GENERATOR OF CARDINALITY M *is* NP*-complete.*

Proof. The problem is in NP. A nondeterministic Turing machine can guess a subset Q of P, and then in polynomial time verify that Q is a generator of P with cardinality no more than m.

To show the hardness, we prove that the NP-complete problem KCM is polynomially reducible to GCM. Let F be a set of FDs on the set of attributes R. To transform this into the corresponding GCM problem, define M to be R, $\mathcal{L}$ to be F, and P to be R. Note that subset Q of P is a generator of P under $\mathcal{L}$ iff it is a key for $< R, F >$. Because if $\mathcal{L}(Q) = P$, then by definition of P, $\mathcal{L}(Q) = R$. Due to the correspondence between implicational closure and closure under FDs, $F \models Q \rightarrow R$, thus Q is a key for $< R, F >$. Now assume that Q is a key for $< R, F >$. Then $F \models Q \rightarrow R$, which means that $\mathcal{L}(Q) = R = P$. Thus Q is a generator of P under $\mathcal{L}$. Obviously, M, $\mathcal{L}$ and P can be determined in time polynomial in $|F|$ and $|R|$. □

Our main motivation to analyze the GCM problem was to see if we can assist the domain expert during attribute exploration by making the implication questions shorter. Above, we have seen that although finding a minimal generating subset of the premise or the conclusion of an implication is can be done in polynomial time, finding a minimum cardinality generator is intractable. So if we ask for an algorithm for this problem, the current state of the art has only worst case exponential answers to offer.

6.3.2 Finding all minimal generators

In some cases, it might not be enough to find only one minimal generator of the premise or conclusion of an implication question. It might be useful to show the expert different minimal generators for better understandability of the implication question. The expert might want to browse among them to find a shortened version of the question that is most comprehensible to him. In the following, we are going to investigate the problem of determining all minimal generators of a closed set.

Above we have mentioned that in [LO78], an algorithm that determines all minimal keys of a given relation was given. Given a set of attributes

R and a set of functional dependencies F, the algorithm returns the set of all minimal keys for the relational system $< R, F >$. Below we are going to give an adaptation of this algorithm to find all minimal generators of a given closed set. The algorithm is based on the following property given in [LO78]. Naturally enough, there the algorithm and the property below are phrased using database terminology. Here we are going to rephrase them in our language of implications and minimal generators of closed sets.

Lemma 6.3.6. *Let $\mathcal{L}$ be a set of implications on M, and $\mathcal{G}$ be a nonempty set of minimal generators for a given $P \subseteq M$ under $\mathcal{L}$. $2^P \setminus \mathcal{G}$ contains a minimal generator iff $\mathcal{G}$ contains a minimal generator G and $\mathcal{L}$ contains an implication $L \to R$ such that $L \cup R \cup G \subseteq P$ and $L \cup (G \setminus R)$ does not include any minimal generator in $\mathcal{G}$.*

Proof. Assume that the condition holds for some G in $\mathcal{G}$ and $L \to R \in \mathcal{L}$. Since $L \cup G \cup R$ includes G and $L \cup G \cup R \subseteq P$, $L \cup G \cup R$ is also a generator. Moreover, $\mathcal{L}(L \cup (G \setminus R)) = L \cup G \cup R$. Thus $L \cup (G \setminus R)$ is a generator; therefore it includes a minimal generator, say G'. Since $L \cup (G \setminus R)$ does not include any minimal generator in $\mathcal{G}$, G' cannot already be in $\mathcal{G}$. That is, the minimal generator G' lies in $2^P \setminus \mathcal{G}$.

Now assume that $2^P \setminus \mathcal{G}$ contains a minimal generator G'. Define G'' to be a maximal subset of P that includes G' but does not include any generator in $\mathcal{G}$. Since $\mathcal{G}$ is nonempty, G'' should be a proper subset of P. Moreover, since G' is a generator, so is G''. Since $G'' \subsetneq P$, and G'' is a generator, $\mathcal{L}$ contains an $L \to R$ such that $G'' \cup R$ is a generator and G'' includes L but does not include R, and $L \cup G'' \cup R \subseteq P$ is satisfied. Since G'' was chosen to be a maximal subset of P that does not include any generator in $\mathcal{G}$, $G'' \cup R$ includes a minimal generator in $\mathcal{G}$, say G. That is, G'' includes both L and $G \setminus R$, therefore it includes $L \cup (G \setminus R)$ and $L \cup G \cup R \subseteq P$ holds. Since G'' does not include any minimal generator in $\mathcal{G}$, neither does $L \cup (G \setminus R)$. □

The lemma assumes a nonempty set of minimal generators, thus the algorithm following from the lemma first needs one minimal generator before it can proceed to find all other minimal generators. We can use Algorithm 6 to obtain the first minimal generator. Based on these, the algorithm that determines the set of all minimal generators of a given closed set is given in Algorithm 7.

The algorithm terminates, since $\mathcal{G}$ and $\mathcal{L}$ are both finite. By Lemma 6.3.6, upon termination $\mathcal{G}$ contains all minimal generators of the given P under $\mathcal{L}$. The algorithm runs in $O(|\mathcal{L}||\mathcal{G}|(|P| + |\mathcal{G}||P|)) + O(|\mathcal{G}|m)$, where m is the complexity of Algorithm 6. That it is, Algorithm 7 has time complexity $O(|\mathcal{L}||\mathcal{G}||P|(|\mathcal{G}| + |P|))$. Note that the algorithm finds minimal generators in *incremental polynomial time*, which is a notion introduced in [JPY88] for analyzing the performance of algorithms that generate all solutions of a problem. An algorithm is said to run in incremental polynomial time, if given

Algorithm 7 All minimal generators

```
1: Initialization
2: L                          {Given set of implications on the attribute set M}
3: P                                          {Given L-closed set P ⊆ M}
4: G := {Minimalgenerator(P, L)}      {Initial set of minimal generators}
5: for all G ∈ G do
6:   for all L → R ∈ L s.t. L ∪ R ∪ G ⊆ P do
7:     S := L ∪ (K \ R)
8:     flag := true
9:     for all H ∈ G do
10:      if H ⊆ S then
11:        flag := false
12:      end if
13:    end for
14:    if flag then
15:      G := G ∪ {Minimalgenerator(S, L)}
16:    end if
17:  end for
18: end for
```

an input and several solutions (say, a closed set and a collection of minimal generators), finds another solution, or determines that none exists, in time polynomial in the combined sizes of the input and the given solutions. For finding a minimal generator, Algorithm 7 needs to do at most $|\mathcal{G}||\mathcal{L}||P|(|\mathcal{G}|+|P|)$ operations, which is polynomial in the size of the input, i.e., in the size of $\mathcal{L}$ and P, and polynomial in the size of the already found minimal generators $\mathcal{G}$.

Another notion introduced in [JPY88] for analyzing algorithms that enumerate solutions, is polynomial delay. An algorithm is said to run with *polynomial delay* if the delay until the first solution is output, and thereafter the delay between any two consecutive solutions is bounded by a polynomial in the size of the input. Polynomial delay is a stronger notion than incremental polynomial time, i.e., if an algorithm runs with polynomial delay it is also runs in incremental polynomial time. To the best of our knowledge, there is no polynomial delay algorithm that finds all minimal keys of a relation, which is equivalent to finding all generators of an attribute set closed under a set of implications.

6.3.3 Counting minimal generators is intractable

In [Osb77], it was shown that the number of minimal keys for a relational system $< R, F >$ can be exponential in $|R|$. This result applies to the number of minimal generators, i.e, the number of minimal generators of a

set closed under a set of implications can be exponential in the size of the attribute set. Below, we show that apart from this exponential bound, it is intractable to determine the number of minimal generators of a set closed under implications. First we define the problem of counting the minimal generators.

Problem 6.3.7. *(#*MINIMAL GENERATOR *of an implication-closed set)*
INSTANCE: A set M of attribute names, a set $\mathcal{L}$ of implications on M, an $\mathcal{L}$-closed subset P of M.
OUTPUT: The number of minimal generators of P under $\mathcal{L}$.

Theorem 6.3.8. #MINIMAL GENERATOR *is #*P*-complete.*

Proof. The problem is in #P. It can be in polynomial time verified that a subset $Q \subseteq P$ is a minimal generator of P under $\mathcal{L}$.

In order to show hardness, we are going to give a parsimonious reduction from the problem of counting all minimal vertex covers of a graph (#MINIMAL VERTEX COVER) to our problem. #MINIMAL VERTEX COVER was shown to be #P-complete in [Val79b]. Recall that in a graph $G = (V, E)$, a set $W \subseteq V$ is a vertex cover of G if for every edge $(v, u) \in E$, $v \in W$ holds, or $u \in W$ holds. W is called a minimal vertex cover if no proper subset of it is a vertex cover. We are going to construct a set of implications and P from an arbitrary graph G, and show that the minimal vertex covers of G are in one-to-one correspondence with the minimal generators of P.

Consider an arbitrary graph $G = (V, E)$. Define the set of attributes as $M = V$, the closed subset of M as $P = V$, and the set of implications as $\mathcal{L} = \{N_v \to \{v\} \mid v \in V, N_v \text{ is the set of vertices adjacent to } v\}$. A subset $W \subseteq V$ is a minimal vertex cover of G iff W is a minimal generator of P under $\mathcal{L}$.

We start with the assertion "if W is a minimal vertex cover, then it is a minimal generator of P under $\mathcal{L}$". If $W = V$, then the assertion holds trivially. Assume $W \subsetneq V$ is a vertex cover of G, and let v be some vertex $v \in V$ such that $v \notin W$. Then the set of vertices N_v that are adjacent to v should be contained in W, i.e. $N_v \subseteq W$. Otherwise W would not be a vertex cover. Then the closure of W under $\mathcal{L}$ contains v, since $\mathcal{L}$ contains the implication $N_v \to \{v\}$, and $N_v \subseteq W$. This means, any $v \in V$, such that $v \notin W$ is contained in $\mathcal{L}(W)$. That is, $\mathcal{L}(W) = V = P$, i.e., W is a generator of P under $\mathcal{L}$. The minimality assertion also holds by the following argument. Let W be a minimal vertex cover of G. Then for any proper subset $W' \subsetneq W$, there is at least one edge $(v, u) \in E$, such that $v \notin W'$ and $u \notin W'$. This means that $N_v \not\subseteq W'$, and thus $v \notin \mathcal{L}(W')$. That is, no proper subset of W generates P, i.e., W is a minimal generator of P.

Now we prove the assertion "if W is a minimal generator of P under $\mathcal{L}$, then it is a minimal vertex cover of G". Again, if $W = P$, the assertion holds trivially. Assume $W \subsetneq P$ is a generator of P under $\mathcal{L}$, and let p be

some attribute $p \in P$ such that $p \notin W$. Since $p \in \mathcal{L}(W)$, and p can only be the consequence of some implication $N_p \to \{p\} \in \mathcal{L}$, $N_p \subseteq W$ should hold. Then for every $p \in P$, $p \in W$ holds or the left-hand side of the implication $N_p \to \{p\} \in \mathcal{L}$ is contained in W, i.e., $N_p \subseteq W$. This means, for every edge $(p, q) \in E$, $p \in W$ holds, or $q \in W$ holds, i.e., W is a vertex cover of G. The minimality assertion also holds by the following argument. Let W be a minimal generator of P. Then for any proper subset $W' \subsetneq W$, there is at least one $p \in P$ such that $p \notin \mathcal{L}(W')$, which means that $N_p \not\subseteq W'$. That is there is at least one edge $(p, q) \in E$ such that $q \in N_p$ and $p \notin W'$ and $q \notin W'$. This means that no proper subset of W is a vertex cover of G, i.e., W is a minimal vertex cover of G.

Obviously, the construction of P and $\mathcal{L}$ can be done in polynomial time, and the reduction is parsimonious, i.e., it preserves the number of solutions. □

6.3.4 Counting minimum cardinality generators is intractable

In this section we consider a modified version of the #MINIMAL GENERATOR problem defined in Problem 6.3.7. For this problem, we slightly change the notion of "generates" as follows: For a given set $\mathcal{L}$ of implications on an attribute set M, and a set $P \subseteq M$ closed under $\mathcal{L}$, in this section we are going to say that a $Q \subseteq M$ is a *minimum cardinality generator* of P if $\mathcal{L}(Q) \setminus Q = P$, and no subset of M with smaller cardinality satisfies this. In other words we require that P should be the "new consequences" of closing Q under $\mathcal{L}$, and that no set with smaller cardinality can have this property. It turns out that the problem of counting such sets is #·coNP-complete, which means that it is even harder than the #MINIMAL GENERATOR problem. First we define the problem formally.

Problem 6.3.9. *(#MINIMUM CARDINALITY GENERATOR)*

INSTANCE: A set M of attribute names, a set $\mathcal{L}$ of implications on M, an $\mathcal{L}$-closed subset P of M.

OUTPUT: The number of all minimum cardinality generators of P under $\mathcal{L}$, i.e., the number of the sets $Q \subseteq M$ such that $\mathcal{L}(Q) \setminus Q = P$ and for no other $R \subseteq M$ with $|R| < |Q|$, $\mathcal{L}(R) \setminus R = P$ holds.

Theorem 6.3.10. #MINIMUM CARDINALITY GENERATOR *is #·coNP-complete.*

Proof. The problem is in #·coNP. This can be shown as follows: given a set of attributes Q, we have to check (i) whether Q generates P, and if so (ii) whether there is another generator R with $|R| < |Q|$. The first check can be done in polynomial time by using the algorithm mentioned in Section 3.3. The second check, which dominates the overall check, can be done by a coNP algorithm. Indeed, checking whether Q is *not* a minimum cardinality generator can be done by the following NP-algorithm: guess a

set $R \subseteq M$ such that $|R| < |Q|$ and check if R generates P. Again checking if R generates P can be done in polynomial time, thus checking whether Q is a minimum cardinality generator can be done in coNP, and counting such sets can be done in #·coNP.

We show #·coNP-hardness by a strong subtractive reduction from the problem #Π_1SAT. As mentioned in Section 6.1.2, #Π_1SAT is #·coNP-complete. Consider an instance of the #Π_1SAT problem given by a formula $\varphi(X) = \forall Y \psi(X, Y)$ where $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_l\}$. W.l.o.g, we can assume that $\psi(X, Y)$ is in 3DNF, i.e., it is of the form $C_1 \vee \ldots \vee C_n$ where each C_i is of the form $C_i = l_{i1} \wedge l_{i2} \wedge l_{i3}$, and the l_{ij}'s are propositional literals over $X \cup Y$.

Let $x'_1, \ldots, x'_k, a_1, \ldots, a_k, y'_1, \ldots, y'_l, b_1, \ldots, b_l, g_1, \ldots, g_n, r$ denote fresh, pairwise distinct variables and let us write them as $X' = \{x'_1, \ldots, x'_k\}$, $Y' = \{y'_1, \ldots, y'_l\}$, $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_l\}$, and $G = \{g_1, \ldots, g_n\}$. We define two instances of the minimum cardinality generator problem. The first problem $\mathbb{P}_1$ is defined as follows:

$$\begin{array}{rcl} M_1 & = & X \cup X' \cup Y \cup Y' \cup A \cup B \cup G \cup \{r\} \\ P_1 & = & A \cup B \cup G \\ \mathcal{L}_1 & = & \{\{x_i, x'_i\} \to M,\ x_i \to a_i,\ x'_i \to a_i \mid 1 \leq i \leq k\} \cup \\ & & \{\{y_i, y'_i\} \to M,\ y_i \to b_i,\ y'_i \to b_i \mid 1 \leq i \leq l\} \cup \\ & & \{z_{ij} \to g_i \mid 1 \leq i \leq n \ \text{ and } \ 1 \leq j \leq 3\} \end{array}$$

where, for $1 \leq s \leq k$ and $1 \leq t \leq l$, z_{ij} is in one of the forms x_s, x'_s, y_t, or, y'_t depending on whether the literal l_{ij} in C_i is in one of the forms $\neg x_s, x_s, \neg y_t$, or, y_t, respectively. In other words, z_{ij} encodes the negation of l_{ij}. Now we define the second problem $\mathbb{P}_2$:

$$\begin{array}{rcl} M_2 & = & M_1 \\ P_2 & = & P_1 \\ \mathcal{L}_2 & = & \mathcal{L}_1 \cup \{\{y_1, \ldots, y_l\} \to g_i \mid 1 \leq i \leq n\} \end{array}$$

Obviously, this construction can be done in polynomial time. Now let $\mathcal{A}(\varphi)$ denote the set of all satisfying truth assignments of a #Π_1SAT-formula φ, and let $\mathcal{B}(\mathbb{P})$ denote the set of all solutions of a minimum cardinality problem $\mathbb{P}$. We claim that the following holds:

$$\mathcal{B}(P_1) \subseteq \mathcal{B}(P_2), \text{ and } |\mathcal{A}(\varphi)| = |\mathcal{B}(\mathbb{P}_2)| - |\mathcal{B}(\mathbb{P}_1)|.$$

Consider the problem $\mathbb{P}_1$. Solutions of $\mathbb{P}_1$, i.e., minimum cardinality generators of P_1 satisfy the following 3 conditions: 1) An attribute a_i can be generated only in two ways, by the implication $x_i \to a_i$, or by the implication $x'_i \to a_i$. So a solution of $\mathbb{P}_1$ contains one of x_i and x'_i. Moreover, it can not contain both of them due to the implication $\{x_i, x'_i\} \to M$, since this implication would also generate r, and r is not contained in P_1. This means, for each $1 \leq i \leq k$ a solution of $\mathbb{P}_1$ contains either x_i or x'_i in order

to be able to generate the a_i's. 2) Similarly, it also contains either y_i or y'_i for each $1 \leq i \leq l$ in order to be able to generate the b_i's. 3) In addition to these, in order to be able to generate an attribute g_i, a solution contains at least one attribute that encodes the negation of a literal occurring in the implicant C_i. For instance, if $C_i = l_{i1} \wedge l_{i2} \wedge l_{i3}$ and $l_{i1} = x_s$, $l_{i2} = y'_t$, and $l_{i3} = x'_u$ where $1 \leq s, u \leq k$ and $1 \leq t \leq l$, then a solution contains at least one of x'_s, y_t, or x_u. In order be able to generate all g_i's, a solution contains at least one such attribute for each implicant C_i. Subsets of M that satisfy these 3 conditions are solutions of $\mathbb{P}_1$. Each such subset has exactly the size $|X| + |Y| = k + l$. Moreover, they are the only solutions of $\mathbb{P}_1$, since any subset of M that has cardinality less than $k + l$ fails to generate at least one attribute in P_1. Conditions 1) and 2) enforce that a solution is a truth assignment over $X \cup Y$. Condition 3) enforces that this truth assignment contains the negation of at least one literal in every implicant, i.e., it enforces that this truth assignment makes the formula $\psi(X, Y)$ *false*.

Now consider the problem $\mathbb{P}_2$. Each solution of $\mathbb{P}_1$ is also a solution of $\mathbb{P}_2$ since $P_2 = P_1$ and $\mathcal{L}_2$ contains all of the implications that $\mathcal{L}_1$ contains. In addition to the implications in $\mathcal{L}_1$, $\mathcal{L}_2$ also contains implications of the form $\{y_1, \ldots, y_l\} \to g_i$ for $1 \leq i \leq n$. These new implications give rise to the following new solutions: Like the solutions of $\mathbb{P}_1$, in order to be able to generate the a_i's and b_i's, they satisfy the conditions 1) and 2) given above. In order to be able to the generate g_i's, they contain every y_i for $1 \leq i \leq l$. In other words, these new solutions are truth assignments over $X \cup Y$ that set every $y_1, \ldots, y_l$ to *true*.

Based on the above descriptions, $\mathcal{B}(\mathbb{P}_1)$ is the set of truth assignments that make $\psi(X, Y)$ *false*, and $\mathcal{B}(\mathbb{P}_2)$ is the set of truth assignments that make $\psi(X, Y)$ *false*, plus the set of truth assignments that set every $y_1, \ldots, y_l$ to *true*. Obviously, the claim $\mathcal{B}(\mathbb{P}_1) \subseteq \mathcal{B}(\mathbb{P}_2)$ is satisfied. Moreover, the difference $\mathcal{B}(\mathbb{P}_1) \setminus \mathcal{B}(\mathbb{P}_2)$ is the set of truth assignments that set every $y_1, \ldots, y_l$ to *true*, and at the same time make $\psi(X, Y)$ *true* (since by taking the set difference from $\mathcal{B}(\mathbb{P}_1)$ we remove the truth assignments that make $\psi(X, Y)$ *false*). In other words, this set contains the models of $\psi(X, Y)$ such that all Y values are fixed by setting them to *true*. This set has exactly the same cardinality as the set of models of $\varphi(X) = \forall Y \psi(X, Y)$, thus the other claim $|\mathcal{A}(\varphi)| = |\mathcal{B}(\mathbb{P}_2)| - |\mathcal{B}(\mathbb{P}_1)|$ is also satisfied. □

6.4 Small generators of concept intents

In this section we consider two problems about generators of concept intents. They are analogs of the problems on generators of implication closed sets, which were worked out in the previous section. As mentioned earlier, by a generator of the intent of a formal concept (A, B) of a formal context $\mathbb{K} = (G, M, I)$, we mean a $Q \subseteq B$ such that $Q'' = B$.

6.4.1 GCM of an intent is intractable

Let us start with the problem of checking whether a concept intent has a generator of cardinality less than a specified size. We call it the GENERATOR OF CARDINALITY M of an intent, or GCM of an intent for short. It is formally defined as follows:

Problem 6.4.1. *(*GENERATOR OF CARDINALITY M *of an intent)*

INSTANCE: A formal context $\mathbb{K} = (G, M, I)$, the intent B of a formal concept (A, B) of $\mathbb{K}$, and a positive integer m.

QUESTION: Is there a subset Q of B of cardinality less than or equal to m that generates B, i.e., is there a $Q \subseteq B$ such that $Q'' = B$ and $|Q| \leq m$?

As in the case of GENERATOR OF CARDINALITY M (or GCM) of a set closed under implications (Problem 6.3.4), this problem is also intractable.

Theorem 6.4.2. GENERATOR OF CARDINALITY M *of an intent is* NP*-complete.*

Proof. Clearly, the problem is in NP. A nondeterministic Turing machine can guess a subset Q of B, and then in polynomial time verify that Q generates B, and has cardinality not more than m.

To show the hardness, we reduce the VERTEX COVER problem to our problem. Recall that in a graph $\mathcal{G} = (V, E)$, a set $W \subseteq V$ is a vertex cover of $\mathcal{G}$ if for every edge $(u, v) \in E$, $u \in W$ holds, or $v \in W$ holds. VERTEX COVER was one of Karp's 21 NP-complete problems in [Kar72].

Consider an arbitrary graph $\mathcal{G} = (V, E)$. We construct a formal context $\mathbb{K} = (G, M, I)$, and a concept intent B from $\mathcal{G}$ as follows: Define $M = V$, $G = E$ and $B = M$, which is the largest concept intent. Define I in the following way: an object $(u, v) \in G$ has an attribute y iff the vertex y is *not* incident to the edge (u, v) in the graph $\mathcal{G}$. More precisely, $(u, v)Iy$ iff y is not incident to (u, v). We claim that a set $W \subseteq V$ is a vertex cover of $\mathcal{G}$ if and only if it generates B under $(\cdot)''$, i.e., $W'' = B$.

We start with the *if* direction of the claim. Assume W is a vertex cover of $\mathcal{G}$. By definitions of the $(\cdot)'$ operator and the incidence relation I, $W' = \{(u, v) \in G \mid \forall w \in W.(u, v)Iw\}$, which is the following set of edges of $\mathcal{G}$: $W = \{(u, v) \in E \mid \neg\exists w \in W.w \textit{ is incident to } (u, v)\}$, i.e., the set of edges that are not incident to any vertex in W. Since we have assumed that W is a vertex cover, there can not be any vertices satisfying the condition in the definition of W'. Thus W' is empty, and W'' is equal to M, i.e., W is a generator of B.

Now we show the *only if* direction of the claim. Assume W is a generator of B, i.e., $W'' = B = M$. Then $W' = \{(u, v) \in G \mid \forall m \in M.(u, v)Im\}$, i.e., set of objects that have all the attributes in M. By the definition of I, this means $W' = \{(u, v) \in E \mid \neg\exists y \in V.y \textit{ is incident to } (u, v)\}$, which is the set of edges that are not incident to any vertex in V. There can not

be any such edges, i.e., W' should be empty, since by definition an edge is incident to exactly two vertices. Then W should be a vertex cover. Because, by definition W' is the set of edges that are not incident to any vertices in W, and it can be empty only if W is a vertex cover. □

6.4.2 Counting minimal generators is intractable

We continue with the problem #MINIMAL GENERATOR of a concept intent, which is the problem of counting minimal generators of a concept intent. We say that a set $Q \subseteq M$ is a *minimal generator of the intent* of a formal concept (A, B) of a formal context $\mathbb{K} = (G, M, I)$, if Q is a generator, i.e., $Q'' = B$, and Q is minimal, i.e., no proper subset of Q has this property. The problem #MINIMAL GENERATOR of a concept intent is formally defined as follows:

Problem 6.4.3. *(#MINIMAL GENERATOR of an intent)*

INSTANCE: A formal context $\mathbb{K} = (G, M, I)$ and the intent B of a formal concept (A, B) of $\mathbb{K}$.

OUTPUT: The number of all distinct minimal generators of B under $(\cdot)''$, i.e., $|\{Q \subseteq M \mid Q'' = B \text{ and } \forall P \subsetneq Q.P'' \neq B\}|$.

As in the case of counting minimal generators of a set closed under implications (Problem 6.3.7), this problem is also intractable.

Theorem 6.4.4. *#MINIMAL GENERATOR of an intent is #P-complete.*

Proof. The problem is in #P. It can be in polynomial time verified that a given $Q \subseteq M$ is a minimal generator of B under $(\cdot)''$.

In order to show the hardness, we are going to give a parsimonious reduction from the #MINIMAL VERTEX COVER problem, which was mentioned in the proof Theorem 6.3.8. In our reduction, we are going to use the same construction used in the proof of Theorem 6.4.2. What we additionally have to show here is that, in this construction minimal vertex covers and minimal generators are in one-to-one correspondence.

Let $\mathcal{G} = (V, E)$ be an arbitrary graph. We construct a formal context $\mathbb{K} = (G, M, I)$ and a concept intent $B \subseteq M$ just like in the proof of Theorem 6.4.2. We define $M = V$, $G = E$, $B = M$ and I as follows: an object $(u, v) \in G$ is in I relation with an attribute y if and only if the vertex y is *not* incident to (u, v), i.e., $(u, v)Iy$ iff y is not incident to (u, v).

Assume $W \subseteq V$ is a vertex cover of $\mathcal{G}$. Then by the same argument in the proof of Theorem 6.4.2, it is a generator of B. Now assume W is minimal. In other words if we remove a vertex from W and call the resulting set X, X will not be a vertex cover, i.e., there will be an edge $(u, v) \in E$ such that $u \notin X$ and $v \notin X$. At the same time, this means that X will not be a generator of B. Because then X' will be $\{(u, v)\}$, and X'' will be $\{u, v\}$.

Thus, if W is a minimal vertex cover, then it is also a minimal generator of B.

In order to prove the *only if* direction of the argument, assume $W \subseteq M$ is a generator of B. Then by the same argument in the proof of Theorem 6.4.2, it is a vertex cover of $\mathcal{G}$. Now assume W is minimal. In other words, if we remove an attribute from W and call the resulting set X, then X will not be a generator of B, i.e, there will be an attribute say $m \in B$, such that $m \notin X''$. This means that X' will contain an object say (u, v) that does not have the attribute m. It also means that X will not be a vertex cover of $\mathcal{G}$. Because X' is the set of edges that are not incident to any of the vertices in X, and this set contains (u, v), i.e., it is not empty. Thus, if W is a minimal generator of B, it is also a minimal vertex cover of $\mathcal{G}$.

Obviously, the construction of G, M, I and B can be done in polynomial time, and the reduction preserves the number of solutions, i.e., it is parsimonious. □

Chapter 7

Conclusion

In the present work, we have presented our contributions to DL research by means of FCA methods, and to the FCA theory itself. Our contributions can be collected under three items.

Supporting bottom-up construction

As a result of extensive use of ontologies in various application domains, there exist now large knowledge bases written in expressive DLs, and modern DL systems that can process them. In order to allow the user to re-use concepts defined in such existing knowledge bases, and still support the bottom-up approach, in [BST04a] an extended bottom-up approach was proposed. In this approach, during the definition of a new concept, the user can use concept names from an existing background knowledge base that is written in an expressive DL $\mathcal{L}$, but can employ a sublanguage of $\mathcal{L}$ for defining concepts. This approach requires to compute the least common subsumer of a set of concept descriptions w.r.t. the background terminology. In the present work, we have presented a practical approximative method based on FCA for computing such common subsumers.

Completing DL knowledge bases

The DL-based ontology language OWL has been recommended by the W3C as the standard ontology language for the semantic web. As a consequence of this standardization, several ontology editors now support OWL, and ontologies written in OWL are employed in more and more applications. As the size of these ontologies grows, tools that support improving their quality becomes an important issue. The tools available until now use DL reasoning to detect inconsistencies and to infer consequences. These approaches address the quality dimension of soundness of an ontology. In the present work, we have worked on a different quality dimension, namely completeness of an ontology. We have provided a basis for formally well-founded techniques and

tools based on FCA that support the ontology engineer in checking whether an ontology contains all the relevant information about some aspects of the application domain, and in extending the ontology appropriately if this is not the case.

On the generators of closed sets

In FCA, so-called closed sets play an important rôle. Concept intents of a formal context are sets closed under the closure operator induced by this formal context. In addition, the sets closed under a set of implications are fundamental to the attribute exploration algorithm. In the present work, we have investigated some decision and counting problems related to the generators of closed sets. We have proven that for both of the above closure operators, checking whether a closed set has a generator of cardinality less than a specified size is intractable, and counting minimal generators is also intractable. In addition, we have shown that counting the minimum cardinality generators of sets closed under the second closure operator mentioned above is also intractable.

7.1 Supporting bottom-up construction

DL-based ontologies are being used in more and more applications in various domains. As a result of this extensive use of ontologies, there exist now large knowledge bases written in expressive DLs, and modern DL reasoners like FaCT [Hor98], RACER [HM01b] and Pellet [SP04] that can reason over these knowledge bases. In order to allow the user to use concepts defined in such existing knowledge bases and still support defining new concepts with the bottom-up approach [BK98, BKM99], in [BST04a] we have proposed an extended bottom-up approach for the following scenario: assume that there is a fixed background terminology defined in an expressive DL, and the user then wants to extend this terminology in order to adapt it to the needs of a particular application domain. However, since the user is not a DL expert, he employs a less expressive DL and needs support through the bottom-up approach when building this user-specific extension of the background terminology. To make this more precise, consider a background terminology (TBox) $\mathcal{T}$ defined in an expressive DL $\mathcal{L}_2$. When defining new concepts, the user employs only a sublanguage $\mathcal{L}_1$ of $\mathcal{L}_2$, for which computing the lcs makes sense. However, in addition to primitive concepts and roles, the concept descriptions written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$.

Supporting the bottom-up approach in this scenario requires the computation of the lcs of $\mathcal{L}_1$ concept descriptions containing names from $\mathcal{T}$. In [BST04b] the case where $\mathcal{L}_1$ is $\mathcal{EL}$ and $\mathcal{L}_2$ is $\mathcal{ALE}$ has been considered and it has been shown that the lcs of such concept descriptions exists. In [BST04a]

this result has been extended to the case where $\mathcal{L}_1$ is $\mathcal{ALE}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. Unfortunately, due to high complexity of the algorithm devised from them, these results can not be used in practice. In order to overcome this problem, we have developed a practical approximative approach based on FCA.

In this approach, we use the attribute exploration method of FCA in order to compute the subsumption hierarchy of conjunctions of concept names (and their negations) defined in the background terminology $\mathcal{T}$. This hierarchy can then be used in the computation of a so-called good common subsumer (gcs), which is not the least common subsumer of the input concepts, but which is more specific than the common subsumers computed by ignoring the terminology $\mathcal{T}$. To this purpose, we have defined a formal context, and shown that the concept lattice of this context is isomorphic to the abovementioned hierarchy. We have also shown that a usual subsumption algorithm for $\mathcal{L}_2$ in the above scenario can act as expert in the attribute exploration during the computation of this hierarchy. We have implemented a first experimental version of the method and run experiments on different knowledge bases. Our experiments showed that the FCA-based method for computing gcs is indeed a successful approach. What we have also noticed in our experiments is that during attribute exploration, the number of counterexamples, thus the size of the formal context gets too big, even for small input terminologies. Later, a more detailed look revealed that due to a specific property of this formal context, namely dichotomy of the attributes, most of these objects are in fact reducible, i.e., they can be removed from the formal context without changing the hierarchy. Based on this, we have developed a new counterexample generation algorithm that only generates irreducible objects so that at the end of attribute exploration, the formal context contains the minimum possible number of objects. We have also implemented this improved algorithm. Our experiments with the new algorithm on the same knowledge bases as above showed that the improved algorithm enables drastic amounts of improvement in the runtime.

We have also shown that from a lattice theoretic point of view, the approaches in [Baa95], [BM00] and our approach are instances of a more general approach, which is computing the lattice of infima of a partially ordered set. We have given a description of this more general approach, and shown that attribute exploration can be used to compute the lattice of the infima of subsets of a partially ordered set as long as the partial order is decidable, and all finite infima are computable.

7.2 Completing DL knowledge bases

The DL-based ontology language OWL has been recommended by W3C as the standard ontology language for the semantic web. As a consequence of this standardization, several ontology editors like Protégé [KFNM04] and

Swoop [KPS+06] now support OWL, and ontologies written in OWL are employed in more and more applications. As the size of these ontologies grows, tools that support improving their quality becomes an important issue. The tools available until now use DL reasoning to detect inconsistencies and to infer consequences. There exist also first approaches that allow to pinpoint the reasons for inconsistencies and for certain consequences. These approaches address the quality dimension of soundness of an ontology. In the present work, we have worked on a different quality dimension, namely completeness of an ontology.

We have described a knowledge acquisition method that allows to extend a DL knowledge base $(\mathcal{T}_0, \mathcal{A}_0)$ by additional information on the relationships that hold in a specific interpretation $\mathcal{I}$ between concepts in a set of concept descriptions M deemed to be interesting. The method extends the TBox of the knowledge base by additional GCIs for relationships that hold in $\mathcal{I}$, but do not follow from the TBox; and it extends the ABox by counterexamples to relationships that do not hold in $\mathcal{I}$, and are not yet refuted by the existing individuals in the ABox. To obtain the necessary information on whether a relationship holds or not, the existing TBox and ABox are checked first. Only if they cannot be used to decide this question, an expert that "knows" $\mathcal{I}$ is asked. The method is based on the well-known attribute exploration approach from FCA. However, this approach had to be extended in order to be able to handle partial contexts, to correctly represent the open-world semantics of DL knowledge bases. There has been some previous work on extending FCA and attribute exploration from complete knowledge to partial knowledge [BH00, BH05]. However, this work is based on assumptions that are different from ours. In order to use attribute exploration method for our setting, we have developed our own extension of FCA and the attribute exploration algorithm. We have shown that the extended algorithm terminates, and it is correct, i.e., the output of the algorithm is a completion of the input knowledge base. This algorithm inherits its complexity from the usual attribute exploration algorithm: in the worst case, which occurs if there are few or many relationships between attributes, it is exponential in the number of attributes. Regarding the number of questions, it asks the expert the minimum number of questions with positive answers. For the questions with negative answers, the behaviour depends on the answers given by the expert: FCA-theory implies that there always exist counterexamples that, if taken in each step, ensure a minimal number of questions with negative answers. In general, however, one cannot assume that the expert provides these "best" counterexamples.

As a formalization of what "relationships between the concepts in M" really means, we have used subsumption relationships between arbitrary conjunctions of elements of M. The reason was, on the one hand, that these relationships should be fairly easy to decide by a domain expert. On the other hand, the close connection between such relationships and implications, as

considered in FCA, facilitated the adaptation of attribute exploration for our purposes. One could also be interested in more complex relationships, however. For example, one could fix a specific description language $\mathcal{L}$ (e.g., comprising some subset of the constructors of $\mathcal{ALC}$), then take as attributes $\mathcal{L}$-concept descriptions over the concept "names" from M, and ask for all subsumption relationships between the conjunctions of the concept descriptions obtained this way. The immediate disadvantage of this extension is that in general the set of attributes is no longer finite, and thus termination of the exploration process is no longer guaranteed. An extension of classical attribute exploration (i.e., for full contexts) in this direction is described in [Rud06]. The main idea to deal with the problem of an infinite attribute set used there is to restrict the attention to concept descriptions with a bounded role depth. But even though this makes the attribute set finite, its size is usually too large for practical purposes. Thus, an adaptation of the method described in [Rud06] to our purposes not only requires an extension of this method to partial contexts, but also some new ideas of how to deal with the practicality issue.

Based on our results, we have implemented a first experimental version of a tool for completing DL knowledge bases as an extension of the ontology editor Swoop [KPS+06], which uses Pellet [SP04] as the underlying DL reasoner. We have just started to evaluate our tool on the OWL ontology for human protein phosphatases mentioned in the introduction, with biologists as experts, and hope to get first significant results on its usefulness and performance in the near future. Unsurprisingly, we have observed that the experts sometimes make errors when answering implication questions. Hence we will extend the completion tool such that it supports detecting such errors and also allows to correct errors without having to restart the exploration from scratch.

7.3 On the generators of closed sets

In FCA, so-called closed sets play an important rôle. For example, the concept intents of a formal context are the sets closed under the closure operator induced by this formal context. In addition, the sets closed under a set of implications are fundamental to the attribute exploration algorithm. In the present work we have investigated a natural problem on closed sets, namely the problem of finding the generators. More precisely, given a closure operator φ on a set G and a set $A \subseteq G$ such that $\varphi(A) = A$, we are interested in finding "small" sets $B \subseteq A$ such that $\varphi(B) = A$. In particular, we have considered the following two closure operators: 1) the closure operator $\mathcal{L}(\cdot)$ induced by a set of implications, and 2) the closure operator $(\cdot)''$ induced by a formal context. By saying small sets we mean the following two notions of minimality: 1) sets minimal w.r.t. set inclusion, 2) sets minimal w.r.t.

cardinality. Based on these two notions of minimality, we have investigated the problem for the two closure operators above.

Solving the first problem, which is the problem of finding small generators of a set closed under implications, can help the expert during attribute exploration by making the implication questions "simpler". The attribute exploration algorithm asks the minimum number of questions to the expert, i.e, none of the questions is redundant. However, it is possible that an implication question can be shortened by removing redundant attributes in its premise and conclusion. We have shown the following results: Given a set of implications $\mathcal{L}$ on M, a set $A \subseteq M$ such that $\mathcal{L}(A) = A$, and a positive integer $m \leq |M|$

- the problem of checking whether A has a generator of cardinality less than or equal to m is NP-complete
- the problem of counting minimal generators of A is #P-complete
- the problem of counting minimum cardinality generators of A is #·coNP-complete

The second problem, which is the problem of finding small generators of a set closed under $(\cdot)''$, is known as finding the minimal generators of a concept intent. Different aspects of it has been considered in the literature [NVRG05, FVG05]. Minimal generators of a concept intent play an important rôle in incremental lattice construction, and lattice merge algorithms. In [NVRG05], the behaviour of minimal generators upon increases in the underlying context's attribute set has been investigated, and a method for computing the family of minimal generators has been presented. Here, we have shown the following results: Given a formal context $\mathbb{K} = (G, M, I)$, a set $A \subseteq M$ such that $A'' = A$, and a positive integer $k \leq |M|$

- the problem of checking if A has a generator of cardinality less than or equal to k is NP-complete
- the problem of counting minimal generators of A is #P-complete

In fact, it is not surprising that the mentioned problems about generators of concept intents and generators of an implication closed set are of the same complexity. Because it is well known that the closure operator induced by a formal context, and the closure operator induced by the set of implications that are valid in this formal context coincide.

Bibliography

[Arm74] W. W. Armstrong. Dependency Structures of Data Base Relationships. In J. L. Rosenfeld, ed., *Proceedings of the Information Processing Congress 74, (IFIP 74)*, pp. 580–583. North-Holland, 1974. ↪ p. 116

[Baa95] F. Baader. Computing a Minimal Representation of the Subsumption Lattice of all Conjunctions of Concepts Defined in a Terminology. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, eds., *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the 1st International KRUSE Symposium*, pp. 168–178. 1995. ↪ p. 14, 21, 22, 61, 75, 76, 77, 131

[Baa03a] F. Baader. Computing the Least Common Subsumer in the Description Logic $\mathcal{EL}$ w.r.t. Terminological Cycles with Descriptive Semantics. In A. de Moor, W. Lex, and B. Ganter, eds., *Proceedings of the 11th International Conference on Conceptual Structures, (ICCS 2003)*, volume 2746 of *Lecture Notes in Computer Science*, pp. 117–130. Springer-Verlag, 2003. ↪ p. 26, 54

[Baa03b] F. Baader. The Instance Problem and the Most Specific Concept in the Description Logic $\mathcal{EL}$ w.r.t. Terminological Cycles with Descriptive Semantics. In A. Günter, R. Kruse, and B. Neumann, eds., *Proceedings of the 26th Annual German Conference on Artificial Intelligence, (KI 2003)*, volume 2821 of *Lecture Notes in Computer Science*, pp. 64–78. Springer-Verlag, 2003. ↪ p. 26, 54

[Baa03c] F. Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In G. Gottlob and T. Walsh, eds., *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 319–324. Morgan Kaufmann, 2003. ↪ p. 26, 54

[Baa03d] F. Baader. Terminological cycles in a description logic with existential restrictions. In G. Gottlob and T. Walsh, eds., *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 325–330. Morgan Kaufmann, 2003. ↪ p. 37

[BBL05a] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope. LTCS-Report LTCS-05-01, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html. ↪ p. 37

[BBL05b] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope. In L. P. Kaelbling and A. Saffiotti, eds., *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, (IJCAI 05)*, pp. 364–369. Professional Book Center, 2005. ↪ p. 37

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ↪ p. 33, 138, 140, 144

[BGSS06] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing Description Logic Knowledge Bases using Formal Concept Analysis. LTCS-Report LTCS-06-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2006. See http://lat.inf.tu-dresden.de/research/reports.html. ↪ p. 32, 88

[BGSS07a] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing Description Logic Knowledge Bases using Formal Concept Analysis. In *Proceedings of the Third International Workshop OWL: Experiences and Directions (OWLED 2007)*. CEUR-WS, 2007. ↪ p. 32, 88

[BGSS07b] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing Description Logic Knowledge Bases using Formal Concept Analysis. In M. M. Veloso, ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 230–235. AAAI Press, 2007. ↪ p. 14, 32, 88

[BH00] P. Burmeister and R. Holzer. On the Treatment of Incomplete Knowledge in Formal Concept Analysis. In B. Ganter and G. W. Mineau, eds., *Proceedings of the 8th International Conference on Conceptual Structures, (ICCS 2000)*, volume 1867

of *Lecture Notes in Computer Science*, pp. 385–398. Springer-Verlag, 2000. ↪ p. 29, 88, 90, 132

[BH05] P. Burmeister and R. Holzer. Treating Incomplete Knowledge in Formal Concept Analysis. In *Formal Concept Analysis*, volume 3626 of *Lecture Notes in Computer Science*, pp. 114–126. Springer-Verlag, 2005. ↪ p. 29, 88, 90, 132

[BHGS01] S. Bechhofer, I. Horrocks, C. A. Goble, and R. Stevens. OilEd: A Reason-able Ontology Editor for the Semantic Web. In F. Baader, G. Brewka, and T. Eiter, eds., *Proceedings of the Joint German/Austrian Conference on AI, (KI 2001)*, Lecture Notes in Computer Science, pp. 396–408. Springer-Verlag, 2001. ↪ p. 38

[BK98] F. Baader and R. Küsters. Computing the Least Common Subsumer and the Most Specific Concept in the Presence of Cyclic $\mathcal{ALN}$-Concept Descriptions. In *Proceedings of the 22nd German Annual Conference on Artificial Intelligence (KI 1998)*, volume 1504 of *Lecture Notes in Computer Science*, pp. 129–140. Springer-Verlag, 1998. ↪ p. 25, 26, 54, 130

[BKM99] F. Baader, R. Küsters, and R. Molitor. Computing Least Common Subsumers in Description Logics with Existential Restrictions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pp. 96–101. 1999. ↪ p. 25, 26, 37, 38, 54, 58, 130

[BKM00] F. Baader, R. Küsters, and R. Molitor. Rewriting Concepts Using Terminologies. In A. Cohn, F. Giunchiglia, and B. Selman, eds., *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pp. 297–308. Morgan Kaufmann Publishers, 2000. ↪ p. 57

[BKT02] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and Difference in Description Logics. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, eds., *Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pp. 203–214. Morgan Kaufmann, 2002. ↪ p. 56, 57

[BL85] R. J. Brachman and H. J. Levesque, eds. *Readings in Knowledge Representation*. Morgan-Kaufmann, Los Altos (CA), USA, 1985. ↪ p. 14, 141, 144, 145

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001. ↪ p. 15, 27, 86

[BM00] F. Baader and R. Molitor. Building and Structuring Description Logic Knowledge Bases Using Least Common Subsumers and Concept Analysis. In B. Ganter and G. W. Mineau, eds., *Proceedings of the 8th International Conference on Conceptual Structures (ICCS 2000)*, volume 1867 of *Lecture Notes in Computer Science*, pp. 292–305. Springer-Verlag, 2000. ↪ p. 14, 21, 23, 79, 84, 131

[BN03] F. Baader and W. Nutt. Basic Description Logics. In *[BCM+03]*, pp. 43–95. 2003. ↪ p. 17

[Bor86] J.-P. Bordat. Calcul pratique du treillis de Galois d' une correspondance. *Mathématiques, Informatique et Sciences Humaines*, 96:31–47, 1986. ↪ p. 42

[Bra04] S. Brandt. Polynomial Time Reasoning in a Description Logic with Existential Restrictions, GCI Axioms, and - What Else? In R. L. de Mántaras and L. Saitta, eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI 2004)*, pp. 298–302. IOS Press, 2004. ↪ p. 37

[Bra06] S. Brandt. *Standard and Non-standard reasoning in Description Logics*. Ph.D. dissertation, Institute for Theoretical Computer Science, TU Dresden, Germany, 2006. ↪ p. 37, 54

[BS01] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001. ↪ p. 61, 76, 77

[BS04] F. Baader and B. Sertkaya. Applying Formal Concept Analysis to Description Logics. In P. Eklund, ed., *Proceedings of the 2nd International Conference on Formal Concept Analysis (ICFCA 2004)*, volume 2961 of *Lecture Notes in Computer Science*, pp. 261–286. Springer-Verlag, Sydney, Australia, 2004. ↪ p. 14, 23, 32, 61, 76, 77

[BST04a] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the Least Common Subsumer w.r.t. a Background Terminology. In J. J. Alferes and J. A. Leite, eds., *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pp. 400–412. Springer-Verlag, Lisbon, Portugal, 2004. ↪ p. 14, 26, 27, 32, 55, 129, 130

[BST04b] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the Least Common Subsumer w.r.t. a Background Terminology. In V. Haarslev and R. Möller, eds., *Proceedings of the 2004*

International Workshop on Description Logics (DL2004), volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, Whistler, Canada, 2004. ↪ p. 32, 57, 130

[BST07] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the Least Common Subsumer w.r.t. a Background Terminology. *Journal of Applied Logic*, 5(3), 2007. ↪ p. 14, 26, 27, 32, 55, 57

[BT01] S. Brandt and A.-Y. Turhan. Using Non-standard Inferences in Description Logics — what does it buy me? In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (KIDLWS'01)*, number 44 in CEUR-WS. RWTH Aachen, 2001. ↪ p. 54

[BVL03] S. Bechhofer, R. Volz, and P. W. Lord. Cooking the Semantic Web with the OWL API. In D. Fensel, K. P. Sycara, and J. Mylopoulos, eds., *Proceedings of the Second International Semantic Web Conference, (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pp. 659–675. Springer-Verlag, 2003. ↪ p. 104

[CAH03] R. Cornet and A. Abu-Hanna. Using Description Logics for Managing Medical Terminologies. In M. Dojat, E. T. Keravnou, and P. Barahona, eds., *Proceedings of the 9th Conference on Artificial Intelligence in Medicine in Europe (AIME 2003)*, volume 2780 of *Lecture Notes in Computer Science*, pp. 61–70. Springer-Verlag, 2003. ↪ p. 67

[CH94] W. W. Cohen and H. Hirsh. Learning the CLASSIC Description Logic: Theoretical and Experimental Results. In *Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pp. 121–133. 1994. ↪ p. 26, 54

[DG84] W. F. Dowling and J. H. Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, 3:267–284, 1984. ↪ p. 44, 46

[DHK05] A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005. ↪ p. 112, 113

[DLN+92] F. M. Donini, M. Lenzerini, D. Nardi, B. Hollunder, W. Nutt, and A. Marchetti-Spaccamela. The Complexity of Existential

Quantification in Concept Languages. *Artificial Intelligence*, 53(2–3):309–327, 1992. ↪ p. 37

[Don03] F. M. Donini. Complexity of Reasoning. In *[BCM+03]*, pp. 96–136. 2003. ↪ p. 37

[FP96] M. Frazier and L. Pitt. CLASSIC Learning. *Machine Learning*, 25(2-3):151–193, 1996. ↪ p. 26, 54

[FRS05] S. Ferré, O. Ridoux, and B. Sigonneau. Arbitrary Relations in Formal Concept Analysis and Logical Information Systems. In F. Dau, M.-L. Mugnier, and G. Stumme, eds., *Proceedings of the 13th International Conference on Conceptual Structures, (ICCS 2005)*, volume 3596 of *Lecture Notes in Computer Science*, pp. 166–180. Springer-Verlag, 2005. ↪ p. 14, 21

[FVG05] C. Frambourg, P. Valtchev, and R. Godin. Merge-Based Computation of Minimal Generators. In F. Dau, M.-L. Mugnier, and G. Stumme, eds., *Proceedings of the 13th International Conference on Conceptual Structures, (ICCS 2005)*, volume 3596 of *Lecture Notes in Computer Science*, pp. 181–194. Springer-Verlag, 2005. ↪ p. 30, 114, 134

[Gan84] B. Ganter. Two Basic Algorithms in Concept Analysis. Technical Report Preprint-Nr. 831, Technische Hochschule Darmstadt, Darmstadt, Germany, 1984. ↪ p. 19, 27, 42, 47, 55, 60

[Gan99] B. Ganter. Attribute Exploration with Background Knowledge. *Theoretical Computer Science*, 217(2):215–233, 1999. ↪ p. 27, 29, 51, 55, 65, 88

[GD86] J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives resultant d'un tableau de données binaries. *Mathématiques, Informatique et Sciences Humaines*, 95:5–18, 1986. ↪ p. 44

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, New York, NY, USA, 1990. ISBN 0716710455. ↪ p. 110, 116

[GK99] B. Ganter and R. Krauße. Pseudo Models and Propositional Horn Inference. Technical Report MATH-AL-15-1999, Institut für Algebra, Technische Universität Dresden, Dresden, Germany, 1999. ↪ p. 27, 55, 65, 69

[GK01] B. Ganter and S. O. Kuznetsov. Pattern Structures and Their Projections. In H. S. Delugach and G. Stumme, eds., *Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, volume 2120 of *Lecture Notes in Computer Science*, pp. 129–142. Springer-Verlag, 2001. ↪ p. 84

[GK05] B. Ganter and R. Krauße. Pseudo-models and propositional Horn inference. *Discrete Applied Mathematics*, 147(1):43–55, 2005. ↪ p. 27, 51, 55, 65, 69

[GMA95] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995. ↪ p. 42

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, Germany, 1999. ↪ p. 19, 40, 41, 60

[Hay79] P. J. Hayes. The Logic of Frames. In D. Metzing, ed., *Frame Conceptions and Text Understanding*, pp. 46–61. Walter de Gruyter and Co., 1979. Republished in [BL85]. ↪ p. 14

[HB91] B. Hollunder and F. Baader. Qualifying Number Restrictions in Concept Languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pp. 335–346. 1991. ↪ p. 67

[HM01a] V. Haarslev and R. Möller. High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In B. Nebel, ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, (IJCAI 2001)*, pp. 161–168. Morgan Kaufmann, 2001. ↪ p. 26, 54

[HM01b] V. Haarslev and R. Möller. RACER system description. In *Proceedings International Joint Conference on Automated Reasoning (IJCAR 2001)*. 2001. ↪ p. 17, 26, 37, 54, 67, 130

[Hol04a] R. Holzer. Knowledge Acquisition under Incomplete Knowledge using Methods from Formal Concept Analysis: Part I. *Fundamenta Informaticae*, 63(1):17–39, 2004. ↪ p. 29, 88, 90

[Hol04b] R. Holzer. Knowledge Acquisition under Incomplete Knowledge using Methods from Formal Concept Analysis: Part II. *Fundamenta Informaticae*, 63(1):41–63, 2004. ↪ p. 29, 88, 90

[Hor98] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proceedings of the 6th International Conference*

on the Principles of Knowledge Representation and Reasoning (KR'98), pp. 636–647. 1998. ↪ p. 17, 26, 37, 54, 130

[HP07] M. Hermann and R. Pichler. Counting Complexity of Propositional Abduction. In M. M. Veloso, ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence, (IJCAI'07)*, pp. 417–422. AAAI Press, 2007. ↪ p. 113

[HPSvH03] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003. ↪ p. 14, 27, 85

[HV95] L. A. Hemaspaandra and H. Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *ACM SIGACT-Newsletter*, 26(1):2–13, 1995. ↪ p. 111

[JPY88] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On Generating All Maximal Independent Sets. *Information Processing Letters*, 27(3):119–123, 1988. ↪ p. 119, 120

[Kar72] R. M. Karp. Reducibility among Combinatorial Problems. In R. Miller and J. Thatcher, eds., *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York, 1972. ↪ p. 125

[KB01] R. Küsters and A. Borgida. What's in an Attribute? Consequences for the Least Common Subsumer. *Journal of Artificial Intelligence Research (JAIR)*, 14:167–203, 2001. ↪ p. 26, 54

[KFNM04] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, eds., *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pp. 229–243. Springer-Verlag, 2004. ↪ p. 27, 38, 86, 131

[KM01a] R. Küsters and R. Molitor. Approximating Most Specific Concepts in Description Logics with Existential Restrictions. In F. Baader, G. Brewka, and T. Eiter, eds., *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI 2001)*, volume 2174 of *Lecture Notes in Computer Science*, pp. 33–47. Springer-Verlag, 2001. ↪ p. 26, 54

[KM01b] R. Küsters and R. Molitor. Computing Least Common Subsumers in $\mathcal{ALEN}$. In *Proceedings of the Seventeenth Inter-*

national Joint Conference on Artificial Intelligence, (IJCAI 2001), pp. 219–224. Morgan Kaufmann, 2001. ↪ p. 26, 54

[KO02] S. O. Kuznetsov and S. A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3):189–216, 2002. ↪ p. 42

[KO06] S. O. Kuznetsov and S. A. Obiedkov. Counting Pseudo-intents and #P-completeness. In R. Missaoui and J. Schmid, eds., *Proceedings of the 4th International Conference on Formal Concept Analysis (ICFCA 2006)*, volume 3874 of *Lecture Notes in Computer Science*, pp. 306–308. Springer-Verlag, Dresden, Germany, 2006. ↪ p. 47

[KPS+06] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, and J. A. Hendler. Swoop: A Web Ontology Editing Browser. *Journal of Web Semantics*, 4(2):144–153, 2006. ↪ p. 27, 86, 104, 132, 133

[KPSG06] A. Kalyanpur, B. Parsia, E. Sirin, and B. C. Grau. Repairing Unsatisfiable Concepts in OWL Ontologies. In Y. Sure and J. Domingue, eds., *The Semantic Web: Research and Applications. Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science*, pp. 170–184. Springer-Verlag, 2006. ↪ p. 27, 86

[Küs01] R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001. Ph.D. thesis. ↪ p. 37

[Kuz93] S. O. Kuznetsov. A fast algorithm for computing all intersections of objects in a finite semi-lattice. *Automatic Documentation and Mathematical Linguistics*, 27(5):11–21, 1993. ↪ p. 42

[Kuz01] S. O. Kuznetsov. On Computing the Size of a Lattice and Related Decision Problems. *Order*, 18(4):313–321, 2001. ↪ p. 42

[Kuz04] S. O. Kuznetsov. On the Intractability of Computing the Duquenne-Guigues Base. *Journal of Universal Computer Science*, 10(8):927–933, 2004. ↪ p. 47

[LO78] C. L. Lucchesi and S. L. Osborn. Candidate Keys for Relations. *Journal of Computer and System Sciences (JCSS)*, 17(2):270–279, 1978. ↪ p. 30, 114, 116, 117, 118, 119

[Lut99] C. Lutz. Complexity of terminological reasoning revisited. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pp. 181–200. Springer-Verlag, 1999. ↪ p. 37

[Lut02] C. Lutz. *The Complexity of Description Logics with Concrete Domains*. Ph.D. dissertation, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002. ↪ p. 37

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Maryland, 1983. ↪ p. 44, 46, 67, 114, 116

[Min81] M. Minsky. A Framework for Representing Knowledge. In J. Haugeland, ed., *Mind Design*, pp. 95–128. The MIT Press, Cambridge (MA), USA, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [BL85]. ↪ p. 14

[Mot06] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. Ph.D. dissertation, Universität Karlsruhe (TH), Germany, 2006. ↪ p. 17, 37

[NB03] D. Nardi and R. J. Brachman. An introduction to Description Logics. In *[BCM+03]*, pp. 1–40. 2003. ↪ p. 17

[Neb90] B. Nebel. Terminological Reasoning is Inherently Intractable. *Artificial Intelligence*, 43(2):235–249, 1990. ↪ p. 36

[Nor78] E. M. Norris. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2):243–250, 1978. ↪ p. 42

[NR99] L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71(5-6):199–204, 1999. ↪ p. 42

[NVRG05] K. Nehmé, P. Valtchev, M. H. Rouane, and R. Godin. On Computing the Minimal Generator Family for Concept Lattices and Icebergs. In B. Ganter and R. Godin, eds., *Proceedings of the Third International Conference on Formal Concept Analysis, (ICFCA 2005)*, volume 3403 of *Lecture Notes in Computer Science*, pp. 192–207. Springer-Verlag, 2005. ↪ p. 30, 114, 134

[Obi02] S. A. Obiedkov. Modal Logic for Evaluating Formulas in Incomplete Contexts. In *Proceedings of the 10th International*

Conference on Conceptual Structures, (ICCS 2002), volume 2393 of *Lecture Notes in Computer Science*, pp. 314–325. Springer-Verlag, 2002. ↪ p. 29, 88

[Osb77] S. L. Osborn. *Normal Forms for Relational Data Bases*. Ph.D. dissertation, University of Waterloo, Canada, 1977. ↪ p. 120

[OVSM04] D. Oberle, R. Volz, S. Staab, and B. Motik. An Extensible Ontology Software Environment. In S. Staab and R. Studer, eds., *Handbook on Ontologies*, International Handbooks on Information Systems, pp. 299–320. Springer-Verlag, 2004. ↪ p. 86

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821. ↪ p. 110

[Pre00] S. Prediger. Terminologische Merkmalslogik in der Formalen Begriffsanalyse. In G. Stumme and R. Wille, eds., *Begriffliche Wissensverarbeitung – Methoden und Anwendungen*, pp. 99–124. Springer-Verlag, Heidelberg, Germany, 2000. ↪ p. 14, 21

[PS99] S. Prediger and G. Stumme. Theory-driven logical scaling: Conceptual information systems meet description logics. In E. Franconi and M. Kifer, eds., *Proceedings of the 6th International Workshop on Knowledge Representation meets Databases (KRDB'99)*. 1999. ↪ p. 14, 21

[Qui67] M. R. Quillian. Word Concepts: A Theory and Simulation of Some Basic Capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [BL85]. ↪ p. 14

[RH97] A. Rector and I. Horrocks. Experience building a large, reusable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*. AAAI Press, 1997. ↪ p. 26, 54

[RHNV07] M. H. Rouane, M. Huchard, A. Napoli, and P. Valtchev. A Proposal for Combining Formal Concept Analysis and Description Logics for Mining Relational Data. In S. O. Kuznetsov and S. Schmidt, eds., *Proceedings of the 5th International Conference on Formal Concept Analysis, (ICFCA 2007)*, volume 4390 of *Lecture Notes in Computer Science*, pp. 51–65. Springer-Verlag, 2007. ↪ p. 14, 21, 22

[Rud03] S. Rudolph. An FCA Method for the Extensional Exploration of Relational Data. In B. Ganter and A. de Moor, eds., *Contributions to International Conference on Conceptual Structures 2003 (ICCS 2003)*, pp. 197–210. Shaker Verlag, 2003. ↪ p. 24

[Rud04] S. Rudolph. Exploring Relational Structures Via $\mathcal{FLE}$. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, eds., *Proceedings of the 12th International Conference on Conceptual Structures (ICCS 2004)*, volume 3127 of *Lecture Notes in Computer Science*, pp. 196–212. Springer-Verlag, 2004. ↪ p. 14, 21, 24

[Rud06] S. Rudolph. *Relational exploration: Combining Description Logics and Formal Concept Analysis for knowledge specification*. Ph.D. dissertation, Fakultät Mathematik und Naturwissenschaften, TU Dresden, Germany, 2006. ↪ p. 14, 21, 24, 29, 88, 133

[SC03] S. Schlobach and R. Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In G. Gottlob and T. Walsh, eds., *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 355–362. Morgan Kaufmann, 2003. ↪ p. 27, 86

[Sch91] K. Schild. A correspondence theory for terminological logics: preliminary report. In J. Mylopoulos and R. Reiter, eds., *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pp. 466–471. Morgan Kaufmann, 1991. ↪ p. 37

[Ser06] B. Sertkaya. Computing the hierarchy of conjunctions of concept names and their negations in a Description Logic knowledge base using Formal Concept Analysis. In *Supplementary Proceedings of the 4th International Conference on Formal Concept Analysis, (ICFCA 2006)*. Dresden, Germany, 2006. ↪ p. 14, 32

[SH00] S. Schulz and U. Hahn. Knowledge Engineering by Large-Scale Knowledge Reuse - Experience from the Medical Domain. In A. G. Cohn, F. Giunchiglia, and B. Selman, eds., *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, pp. 601–610. Morgan Kaufmann, 2000. ↪ p. 26, 54

[SP04] E. Sirin and B. Parsia. Pellet: An OWL DL Reasoner. In *Proceedings of the 2004 International Workshop on Description*

Logics (DL2004), volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004. ↪ p. 17, 26, 37, 54, 130, 133

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991. ↪ p. 15, 37

[STB+00] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Fast Computation of Concept lattices Using Data Mining Techniques. In M. Bouzeghoub, M. Klusch, W. Nutt, and U. Sattler, eds., *Proceedings of the 7th International Workshop on Knowledge Representation Meets Databases (KRDB 2000)*, volume 29 of *CEUR Workshop Proceedings*, pp. 129–139. CEUR-WS.org, Berlin, Germany, 2000. ↪ p. 42

[Stu96a] G. Stumme. Attribute Exploration with Background Implications and Exceptions. In H.-H. Bock and W. Polasek, eds., *Data Analysis and Information Systems. Statistical and Conceptual approaches. Proceedings of GfKl'95. Studies in Classification, Data Analysis, and Knowledge Organization 7*, pp. 457–469. Springer-Verlag, 1996. ↪ p. 65

[Stu96b] G. Stumme. The Concept Classification of a Terminology Extended by Conjunction and Disjunction. In N. Y. Foo and R. Goebel, eds., *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence (PRICAI'96)*, volume 1114 of *Lecture Notes in Computer Science*, pp. 121–131. Springer-Verlag, 1996. ↪ p. 14, 21, 23

[Stu98] G. Stumme. Distributive Concept Exploration - A Knowledge Acquisition Tool in Formal Concept Analysis. In O. Herzog and A. Günter, eds., *Proceedings of the 22nd Annual German Conference on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pp. 117–128. Springer-Verlag, 1998. ↪ p. 23

[SYvWM04] G. Schopfer, A. Yang, L. von Wedel, and W. Marquardt. CHEOPS: A tool-integration platform for chemical process modelling and simulation. *International Journal on Software Tools for Technology Transfer*, 6(3):186–202, 2004. ↪ p. 67

[TH06] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In U. Furbach and N. Shankar, eds., *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pp. 292–297. Springer-Verlag, 2006. ↪ p. 17, 26, 37

[TK04a] A.-Y. Turhan and C. Kissig. Sonic - Non-standard Inferences Go OilEd. In D. A. Basin and M. Rusinowitch, eds., *Proceedings of the Second International Joint Conference, (IJCAR 2004)*, Lecture Notes in Computer Science, pp. 321–325. Springer-Verlag, 2004. ↪ p. 38

[TK04b] A.-Y. Turhan and C. Kissig. Sonic: System Description. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004. ↪ p. 38

[Tod91] S. Toda. *Computational Complexity of Counting Complexity Classes*. Ph.D. dissertation, Tokyo Institute of Technology, Department of Computer Science, 1991. ↪ p. 112

[Tur07] A.-Y. Turhan. *On the Computation of Common Subsumers in Description Logics*. Ph.D. dissertation, Institute for Theoretical Computer Science, TU Dresden, Germany, 2007. In preparation. ↪ p. 57, 60, 70

[Val79a] L. G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. ↪ p. 110, 111

[Val79b] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. ↪ p. 110, 111, 121

[VM01] P. Valtchev and R. Missaoui. Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods. In H. S. Delugach and G. Stumme, eds., *Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, volume 2120 of *Lecture Notes in Computer Science*, pp. 290–303. Springer-Verlag, Stanford, CA, USA, 2001. ↪ p. 42

[WBH+05] K. Wolstencroft, A. Brass, I. Horrocks, P. W. Lord, U. Sattler, D. Turi, and R. Stevens. A Little Semantic Web Goes a Long Way in Biology. In *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pp. 786–800. Springer-Verlag, 2005. ↪ p. 28, 86

[Wil82] R. Wille. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In I. Rival, ed., *Ordered Sets*, pp. 445–470. Reidel, Dordrecht-Boston, 1982. ↪ p. 17, 28, 87

[Zic91] M. Zickwolff. *Rule Exploration: First Order Logic in Formal Concept Analysis*. Ph.D. dissertation, TH Darmstadt, Germany, 1991. ↪ p. 14, 21

Index