# Techniques for Ontology Design and Maintenance

## Deliverable TONES-D13

F. Baader[4], R. Bernardi[1], D. Calvanese[1], A. Calì[1], B. Cuenca Grau[3], M. Garcia[5], G. de Giacomo[2], A. Kaplunova[5], O. Kutz[3], D. Lembo[2], M. Lenzerini[2], L. Lubyte[1], C. Lutz[4], M. Milicic[4], R. Möller[5], B. Parsia[3], R. Rosati[2], U. Sattler[3], B. Sertkaya[4], S. Tessaris[1], C. Thorne[1], and A.-Y. Turhan[4]

[1] Free University of Bozen-Bolzano
[2] Università di Roma "La Sapienza"
[3] The University of Manchester
[4] Technische Universität Dresden
[5] Technische Universität Hamburg-Harburg


| | |
|---|---|
| Project: | FP6-7603 – Thinking ONtologiES (TONES) |
| Workpackage: | WP3 – Ontology Design and Maintenance |
| Lead Participant: | Technische Universität Dresden |
| Reviewer: | Ralf Möller |
| Document Type: | Deliverable |
| Classification: | Public |
| Distribution: | TONES Consortium |
| Status: | Final |
| Document file: | D13_TechniquesDesign.pdf |
| Version: | 3.0 |
| Date: | January 30, 2007 |
| Number of pages: | 187 |

| **Document Change Record** | | |
|---|---|---|
| **Version** | **Date** | **Reason for Change** |
| v.0.1 | December 4, 2006 | Outline |
| v.1.0 | January 15, 2007 | First complete version |
| v.1.1 | January 15, 2007 | Corrections made |
| v.2.0 | January 22, 2007 | Version sent to reviewer |
| v.2.1 | January 26, 2007 | Reviewer comments incorporated |
| v.3.0 | January 30, 2007 | Final version |

# Contents

**1 Introduction** **6**

**2 Description Logics** **8**
2.1 Syntax and Semantics of $\mathcal{SHOIQ}$ . . . . . 8

**3 Tasks in Ontology Design and Maintenance** **11**
3.1 Authoring Concept Descriptions . . . . . 11
3.2 Generating Concept Descriptions . . . . . 11
3.3 Structuring the Ontology . . . . . 11
3.4 Bottom-up Construction . . . . . 11
3.5 Stepwise Extension . . . . . 11
3.6 Ontology Customization . . . . . 12
3.7 Concept Inspection . . . . . 12
3.8 Error management . . . . . 12
3.9 View-based Ontology Design and Maintenance . . . . . 12
3.10 Ontologies from Database Schema . . . . . 12

**4 Reasoning in Lightweight DLs** **13**
4.1 Introduction . . . . . 13
4.2 Description Logics . . . . . 14
4.3 Deciding Subsumption in $\mathcal{EL}^{++}_{\mathrm{rr}}$ and $\mathcal{EL}^{++}_{\mathrm{ri}}$ extended by concrete domains . 16
4.3.1 A Normal Form for CBoxes . . . . . 17
4.3.2 Eliminating Range Restrictions . . . . . 18
4.3.3 The Algorithm . . . . . 19
4.4 P-admissible and Non-admissible Concrete Domains . . . . . 22
4.5 Lower Bounds . . . . . 23
4.5.1 Undecidability of Full $\mathcal{EL}^{++}$ . . . . . 24
4.5.2 Atomic negation . . . . . 24
4.5.3 Disjunction . . . . . 25
4.5.4 At-Least Restrictions . . . . . 25
4.6 Related Work . . . . . 26

**5 Reasoning in OWL 1.1 and $\mathcal{SROIQ}$** **27**
5.1 Introduction . . . . . 27
5.2 The Description Logic $\mathcal{SROIQ}$ . . . . . 28
5.3 Concepts and Inference Problems for $\mathcal{SROIQ}$ . . . . . 32
5.4 Reduction of Inference Problems . . . . . 33
5.5 $\mathcal{SROIQ}$ is Decidable . . . . . 35
5.5.1 A Tableau for $\mathcal{SROIQ}$ . . . . . 36
5.5.2 The Tableau Algorithm . . . . . 38
5.6 Outlook . . . . . 41

**6 Description Logics with Concrete Domains 43**
6.1 Introduction . . . 43
6.2 Constraint Systems . . . 44
6.2.1 RCC8 . . . 45
6.2.2 Allen's Relations . . . 46
6.2.3 Properties of Constraint Systems . . . 47
6.3 Syntax and Semantics . . . 48
6.4 A Tableau Algorithm for $\mathcal{ALC}(\mathcal{C})$ . . . 50
6.4.1 Normal Forms . . . 50
6.4.2 Data Structures . . . 51
6.4.3 The Tableau Algorithm . . . 53
6.5 Practicability . . . 56
6.6 Related Work . . . 57

**7 Pinpointing in Expressive DLs 58**
7.1 Introduction . . . 58
7.2 Justification of Entailments and MUPS . . . 59
7.3 Finding Justifications . . . 60
7.3.1 Computing a Single Justification . . . 61
7.3.2 Computing All Justifications . . . 62
7.4 Error Repair . . . 63
7.4.1 Strategies for Ranking Axioms . . . 64
7.4.2 Generating Repair Solutions . . . 67
7.4.3 Suggesting Axiom Rewrites . . . 69
7.5 Glass-box algorithms . . . 70
7.6 Related Work . . . 72

**8 Model Checking and Model Generation 74**
8.1 Introduction . . . 74
8.2 Translation from Description Logics into Alloy . . . 76
8.2.1 Translation Rules for $\mathcal{ALC}$ . . . 76
8.2.2 Translation of $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$ . . . 78
8.2.3 Translation Algorithm . . . 79
8.3 Case Studies . . . 80
8.3.1 Model Inspection by Counterexample Extraction . . . 80
8.3.2 Counterexamples for a Subsumption Assumption . . . 82
8.3.3 Counterexamples for a Concept Equivalence Assumption . . . 84
8.4 Evaluation of Practical Usefulness . . . 85
8.5 Conclusion and Further Work . . . 86

**9 Conservative Extensions 88**
9.1 Introduction . . . 88
9.2 Preliminaries . . . 88
9.3 Conservative Extensions in $\mathcal{ALC}$ . . . 89
9.4 Model-Conservative Extensions in $\mathcal{ALC}$ . . . 91

9.5 Conservative Extensions in $\mathcal{ALCQI}$ . . . 92
9.6 Conservative Extensions in $\mathcal{ALCQIO}$ . . . 96
9.7 Related Work . . . 97

**10 FCA-based Completion of TBoxes 98**
10.1 Introduction . . . 98
10.2 Formal Concept Analysis . . . 99
10.3 Partial contexts . . . 103
10.4 Attribute exploration with partial contexts . . . 105
10.5 DLs and partial contexts . . . 107
10.6 Completion of DL knowledge bases . . . 109
10.7 Related Work . . . 111

**11 Computing Common Subsumers 113**
11.1 Introduction . . . 113
11.2 Preliminaries . . . 114
11.3 Existence and non-existence of the lcs w.r.t. TBoxes . . . 117
11.3.1 Existence and non-existence of the $\mathcal{EL}(\mathcal{T})$-lcs . . . 117
11.3.2 Existence and non-existence of the $\mathcal{ALE}(\mathcal{T})$-lcs . . . 118
11.4 Good common subsumers . . . 119
11.4.1 The lcs of $\mathcal{ALE}$-concept descriptions . . . 119
11.4.2 A good common subsumer in $\mathcal{ALE}$w.r.t. a background TBox . . . 120
11.4.3 Using $\mathcal{ALE}$-expansion when computing the gcs . . . 120
11.4.4 Alternative approaches for computing common subsumers . . . 122
11.5 Computing the subsumption lattice of conjunctions of (negated) concept names w.r.t. a TBox . . . 124
11.6 Future and Related Work . . . 125

**12 Ontology Extraction from DB Schemas 126**
12.1 Introduction . . . 126
12.2 Preliminaries . . . 126
12.2.1 Relational Model . . . 126
12.2.2 Ontology Language . . . 127
12.2.3 Relational Schemata . . . 129
12.3 Principles of conceptual schema extraction . . . 131
12.4 Conceptual schema extraction algorithms . . . 133
12.5 Related Work . . . 137

**13 Query Containment and Satisfiability 138**
13.1 Introduction . . . 138
13.2 Checking Query Containment . . . 143
13.2.1 Containment by chasing . . . 143
13.2.2 Decidabilty of containment . . . 144
13.3 Related work . . . 146

**14 Query Satisfiability, Disjointness, and Containment 148**
14.1 Introduction . . . 148
14.2 DL-Lite . . . 149
14.3 Query Satisfiability . . . 151
14.3.1 An algorithm for KB satisfiability . . . 151
14.3.2 An algorithm for query satisfiability . . . 153
14.4 Query Disjointness . . . 155
14.5 Query Containment . . . 155
14.6 Related Work . . . 158

**15 Ontology specification in natural language 159**
15.1 Introduction . . . 159
15.2 The framework . . . 159
15.3 *DL-Lite* and Natural Language . . . 160
15.3.1 Introduction to *DL-Lite* . . . 160
15.3.2 Fragment of Natural Language for *DL-Lite* . . . 161
15.4 CG and Natural Language for *DL-Lite* . . . 163
15.4.1 Introduction to Categorial Grammar . . . 163
15.4.2 *CG-Lite* . . . 165
15.5 Related Work . . . 167

**References 169**

# 1 Introduction

Workpackage 3 of the TONES project is concerned with automated reasoning support of ontology design and maintenance. As discussed in much more detail in the previous deliverable [LLS06], such support is needed to assist the ontology designer in carrying out a principled and systematic design-process, and to ensure that the resulting ontology is well-structured and useful for the intended application. To maintain the high-quality structure throughout the whole ontology life-cycle, it is additionally necessary to carry out maintenance and evolution in a structured and systematic way. Also here, support provided by automated reasoning tools can be of tremendous benefit.

In the deliverable [LLS06], we have identified and described the tasks that play a crucial role during ontology design and maintenance. We have also identified services that should be offered by ontology design tools and methodologies to support the ontology engineer in the identified tasks. It turned out that, often, these services are closely related to reasoning problems that have been studied in logic. Therefore, in [LLS06] we have also identified and discussed logical reasoning problems that can be used to implement the desirable services for ontology design and maintenance. Finally, we have given a brief survey of the available results and techniques, and identified important open research issues.

The purpose of the current deliverable is to summarize the techniques that realize the reasoning services identified as fundamental for ontology design and maintenance, and to report on their computational properties. In particular, we concentrate on novel techniques and results that have been developed within the TONES project. To discuss logical reasoning techniques in detail, we need to select a concrete logical formalism that is used as an ontology language. For the remainder of this deliverable, the selected formalism will be description logic (DL). A good introduction to DL and a comprehensive overview of the field is provided by the handbook [BCM+03b]. The use of DLs as ontology languages has been described for example in the survey articles [BHS02] and [BHS03a]. We have chosen DLs as the ontology language for the following two reasons:

1. Description logics currently have a very strong standing as ontology languages. Most notably, under the names OWL-Lite and OWL-DL they have been standardized as the ontology language of the web. A description of the history and development of the OWL standard can be found in [HPSvH03]; see also Section 5.

2. The core definitions of description logics capture the core aspects of ontologies in a straightforward and uncontroversial way. Since we will mainly use the core of DLs in this deliverable, most of the discussion of DL reasoning problems provided in this section applies also to other ontology languages.

The deliverable starts with a brief introduction to description logics in Section 2. In Section 3, we briefly summarize the tasks for ontology design and maintenance identified in Deliverable [LLS06] for later reference. Each of the following technical sections concentrates on a family of techniques that have been developed within TONES. For example, Section 5 discusses reasoning techniques for the new 1.1 version of the ontology language OWL and the closely related description logic $\mathcal{SROIQ}$. Both OWL 1.1 and $\mathcal{SROIQ}$,

as well as the presented reasoning techniques, have been developed within the TONES project. All the technical sections have an identical structure: we first give a brief introduction to the addressed problem, relating it to the ontology design tasks in Section 3 and to the common logical framework developed in Deliverable [CGG+06]. Then the techniques and results are presented in detail. Finally, we put these results into perspective of related and existing work.

# 2 Description Logics

In this section, we introduce the description logic $\mathcal{SHOIQ}$ and its fragments. We have chosen this particular DL for presentation at this point because $\mathcal{SHOIQ}$ forms the basis of the OWL 1.0 standard. Additionally, it is very expressive and most of the DLs referred to in this deliverable can be conveived as fragments of $\mathcal{SHOIQ}$.

## 2.1 Syntax and Semantics of $\mathcal{SHOIQ}$

In DLs, *concepts* and *roles* are the main ingredients to the concept definitions that comprise an ontology (also called *knowledge base* in DL parlance). Concepts and roles are inductively defined with the help of a set of *constructors*, starting with a set $\mathsf{N_C}$ of *concept names*, a set $\mathsf{N_R}$ of *role names*, and (possibly) a set $\mathsf{N_I}$ of *individual names*. In some DLs such as $\mathcal{SHOIQ}$, the set of role names contains a subset $\mathsf{N_{tR}} \subseteq \mathsf{N_R}$ of *transitive role names*. In $\mathcal{SHOIQ}$, there is only a single role constructor: a *role* is an element of $\mathsf{N_R} \cup \{r^- \mid r \in \mathsf{N_R}\}$, where roles of the form $r^-$ are called *inverse roles*.

*$\mathcal{SHIQ}$-concepts* (or concepts for short) are built inductively using the following grammar, where $A \in \mathsf{N_C}$, $a \in \mathsf{N_I}$, $n \geq 0$, $r$ is a role, and $s$ is a role such that neither $s \in \mathsf{N_{tR}}$ nor $s = r^-$ for some $r \in \mathsf{N_{tR}}$:

$$C ::= \top \mid \bot \mid A \mid \{a\} \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall r.C \mid \exists r.C \mid (\leqslant n\ s\ C) \mid (\geqslant n\ s\ C)$$

The concepts $\top$ and $\bot$ denote logical truth and falsity. Concepts of the form $\{a\}$ are called *nominals*, concepts of the form $\forall r.C$ *universal restrictions*, and concepts of the form $\exists r.C$ *existential restrictions*. Finally, concepts of the form $(\leqslant n\ r\ C)$ and $(\geqslant n\ r\ C)$ are called (qualified) *number restrictions*.

The semantics for concepts and roles are provided by *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consist of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, which maps every concept name $A$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every role name $r \in \mathsf{N_R}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that $r^{\mathcal{I}}$ is transitive for every $r \in \mathsf{N_{tR}}$, and every individual name $a$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. This semantics is extended to complex concepts and roles as follows:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{ if } (d,d') \in r^{\mathcal{I}}, \text{ then } d' \in C^{\mathcal{I}}\} \\
(\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{ there is } d' \in C^{\mathcal{I}} \text{ with } (d,d') \in r^{\mathcal{I}}\} \\
(\leqslant n\ s\ C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \#\{d' \in C^{\mathcal{I}} \mid (d,d') \in s^{\mathcal{I}}\} \leqslant n\} \\
(\geqslant n\ s\ C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \#\{d' \in C^{\mathcal{I}} \mid (d,d') \in s^{\mathcal{I}}\} \geqslant n\}
\end{aligned}$$

where $\#S$ denotes the cardinality of the set $S$.

A $\mathcal{SHOIQ}$ ontology consists of three components, which we introduce next. The first component is a *TBox*, which is a finite set of *general concept inclusions (GCIs)* of the form $C \sqsubseteq D$, where $C$ and $D$ are $\mathcal{SHOIQ}$ concepts. The second component is a *role hierarchy*, which is a finite set of *role inclusions* $r \sqsubseteq s$, where $r$ and $s$ are roles. Finally, the third component is an *ABox*, which is a finite set of *assertions* of the form $C(a)$ or $r(a, b)$, where $C$ is a concept, $r$ a role, and $a, b \in \mathsf{N_I}$. Thus, a $\mathcal{SHOIQ}$ *ontology* is a triple $(\mathcal{T}, \mathcal{H}, \mathcal{A})$ with $\mathcal{T}$ a TBox, $\mathcal{H}$ a role hierarchy, and $\mathcal{A}$ an ABox. In this deliverable, we will also call such a triple a *knowledge base* and sometimes refer to the TBox component alone as the ontology.

As for the semantics, an interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, a role inclusion $r \sqsubseteq s$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$, an assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and an assertion $r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. It satisfies a TBox/role hierarchy/ABox if it satisfies all GCIs/role inclusions/assertions in it.

When we use an ontology with a non-empty role hierarchy, we have to strenghten our assumption on the roles allowed inside number restrictions $(\leqslant n\ s\ C)$ and $(\geqslant n\ s\ C)$. Recall that we disallowed transitive roles there, i.e., demanded that neither $s \in \mathsf{N_{tR}}$ nor $s = r^-$ for some $r \in \mathsf{N_{tR}}$. In the presence of role hierarchies, there may be different reasons for a role to be transitive. This is captured by the notion of a simple role, defined as follows:

- $\mathsf{Inv}(r) := r^-$ if $r \in \mathsf{N_R}$ and $\mathsf{Inv}(r) := s$ if $r = s^-$ for a role name $s$.
- For a role hierarchy $\mathcal{H}$, $\sqsubseteq^*_{\mathcal{H}}$ is the reflexive transitive closure of $\sqsubseteq$ over $\mathcal{H} \cup \{\mathsf{Inv}(r) \sqsubseteq \mathsf{Inv}(s) \mid r \sqsubseteq s \in \mathcal{H}\}$, and we use $r \equiv^*_{\mathcal{H}} s$ as an abbreviation for $r \sqsubseteq^*_{\mathcal{H}} s$ and $s \sqsubseteq^*_{\mathcal{H}} r$.
- For a role hierarchy $\mathcal{H}$ and a role $s$, we define the set $\mathsf{Trans}_{\mathcal{H}}$ of transitive roles as

$$\{s \mid \exists \text{ role } r \text{ with } r \equiv^*_{\mathcal{H}} s \text{ and } r \in \mathsf{N_{tR}} \text{ or } \mathsf{Inv}(r) \in \mathsf{N_{tR}}\}.$$

- A role $r$ is called *simple* w.r.t. a role hierarchy $\mathcal{H}$ if, for each role $s$ such that $s \sqsubseteq^*_{\mathcal{H}} r$, $s \notin \mathsf{Trans}_{\mathcal{H}}$.

This finishes the introduction of $\mathcal{SHOIQ}$. In the remainder of this section, we identify a number of important fragments. We start with introducing to rather basic DLs. First, the DL that

1. has no role constructors (and thus no inverse roles) and no role hierarchies;
2. has only the concept constructors negation, conjunction, disjunction, and universal and existential restriction;
3. does not have transitive roles

is called $\mathcal{ALC}$. And second the DL that extends $\mathcal{ALC}$ with role hierarchies and transitive roles is called $\mathcal{S}$.

The availability of additional constructors in $\mathcal{ALC}$ and $\mathcal{S}$ is indicated by concatenation of a corresponding letter: $\mathcal{Q}$ stands for number restrictions; $\mathcal{I}$ stands for inverse roles, and

$\mathcal{O}$ for nominals. Thus, $\mathcal{ALCO}$ is $\mathcal{ALC}$ extended with nominals, $\mathcal{ALCQIO}$ is the fragment of $\mathcal{SHOIQ}$ that does not allow role hierarchies and transitive roles, and $\mathcal{SHIQ}$ is the fragment of $\mathcal{SHOIQ}$ that does not allow nominals.

The main task of a DL system is to infer implicit knowledge from the knowledge base and make it explicit. The most relevant inference problems can be described as follows:

- *Concept satisfiability.* A concept $C$ is *satisfiable w.r.t. a knowledge base* $\mathcal{K}$ iff there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$.
- *Concept subsumption.* A concept $C$ *subsumes* a concept $D$ w.r.t. a knowledge base $\mathcal{K}$ (written $C \sqsubseteq_{\mathcal{K}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{K}$.
- *Knowledge base consistency.* A knowledge base $\mathcal{K}$ is *consistent* iff $\mathcal{K}$ has a model.
- *The instance problem.* An individual name $a$ is an *instance* of a concept $C$ in a knowledge base $\mathcal{K}$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

# 3 Tasks in Ontology Design and Maintenance

The main purpose of deliverable D05 was to identify design and maintenance tasks that are crucial for achieving and preserving a high-quality formalization of knowledge in the ontology. Here, we repeat a short summary of each identified task. These summaries will be used as reference points in the discussion of techniques for supporting the identified tasks, which comprises the main part of the current deliverable. For more details on the tasks listed in the following, we refer to deliverable D05 [LLS06].

## 3.1 Authoring Concept Descriptions

The author wants to add a new concept description to the ontology or modify a concept description that was already contained in the ontology. This may happen either in the design phase of the ontology or during the maintenance phase. After producing a candidate description of the concept, the author needs to understand the implicit consequences of his modelling and the interaction of this description with the other descriptions in the ontology.

## 3.2 Generating Concept Descriptions

The ontology designer wants to add a new concept to the ontology, but finds it difficult to describe it. To obtain a starting point for the concept description, the designer wants to automatically generate an initial description of the new concept that is based on the position of this concept in the subsumption hierarchy.

## 3.3 Structuring the Ontology

The ontology designer wants to improve the structure of an ontology by inserting intermediate concepts into the subsumption hierarchy. He needs support to decide where to add such concepts and how to describe them.

## 3.4 Bottom-up Construction

The ontology designer wants to design the ontology bottom-up, i.e., by proceeding from the most specific concepts to the most general ones. This should be supported by automatically generating concept descriptions from descriptions of typical instances of the new concept (and, additionally, of intended subclasses in the hierarchy–if any).

## 3.5 Stepwise Extension

To develop an ontology, the designer starts with formalizing only a part of the domain, and then extends the ontology with notions from additional parts in a stepwise fashion. When adding concept descriptions from new parts of the application domain to the existing ontology, he wants to avoid that the existing and debugged ontology is compromised by the extension.

## 3.6 Ontology Customization

An ontology user wants to adapt an existing ontology to his purposes by making simple modifications. Since he is not an expert in ontology languages, he works with a simpler language than the one used to formulate the ontology and/or with graphical frame-like interfaces.

## 3.7 Concept Inspection

The ontology designer wants to display a concept description in a way that facilitates understanding of the concept's meaning. He wants to query the ontology for concepts that are similar to a given concept description.

## 3.8 Error management

An automated reasoning tool has reported a problem in the ontology. The ontology designer wants support in pinpointing the source of the problem, have the problem explained in an understandable way, and/or have suggestions on how to resolve the problem.a

## 3.9 View-based Ontology Design and Maintenance

In designing and maintaining an ontology, the developer may want to use views (i.e., queries) specified over the ontology to have a result of high quality. She/he may have some typical views at hand and wants to test whether they are non-empty and whether some of which are contained in some others. Based on the knowledge about the domain of the ontology developer, he could conclude that the ontology contains flaws if the results are not as he would expect. We point out that this task comprises and refines the "Query Management" task introduced in the previous deliverable D05 [LLS06].

## 3.10 Ontologies from Database Schema

The ontology designer wants to create an ontology about an enterprise domain that is already (partly) described in the form of a database schema. Instead of constructing the ontology from scratch, he wants automatic support for converting the database schema into an initial ontology.

# 4 Reasoning in Lightweight DLs

## 4.1 Introduction

When designing an ontology language, one faces the ubiquitous tradeoff between the expressive power of the language and the computational complexity of reasoning in it. On the one hand, greater expressive power allows to describe the relevant concepts of the application domain in more detail. On the other hand, higher computational complexity means that reasoning may become infeasible if ontologies become larger or answer time plays a crucial role.

There is no one optimal solution to this tradeoff. In particular, it is not possible to find an optimal solution without taking into consideration the application for which the ontology is designed. There exist applications that require a rather detailed modelling of concepts in the ontology, and in which ontologies are small enough so that reasoning in computationally expensive languages is possible. Applications of this kind may be using onology languages such as those discussed in Section 5. In other applications such as medical informatics and genomics, ontologies tend to be extremely large and can usually not be processed by reasoners that support expressive and computationally expensive ontology languages. Fortunately, it seems to be the case that applications of the latter kind often only need a much more abstract modelling of the relevant concepts, and thus a relatively inexpressive ontology language is sufficient.

In this section, we explore description logics for which reasoning w.r.t. general TBoxes (i.e., sets of GCIs) is tractable and that are still sufficiently expressive to be used in relevant applications such as medical informatics, bioinformatics, and genomics. More precisely, we focus on the most important reasoning problem during ontology design, namely TBox *classification*: given a TBox $\mathcal{T}$, compute the subsumption relation between all concept names that appear in $\mathcal{T}$. Classification is of crucial importance for almost all ontology design task. In particular, it plays a central role for authoring concept constructions and structuring the ontology. From the perspective of the common logical framework in deliverable D08, we are concerned with stand-alone ontologies.

The quest for tractable (i.e., polynomial-time decidable) description logics (DLs) started in the 1980s after the first intractability results for DLs were shown [BL84, Neb88]. Until recently, it was restricted to DLs that extend the basic language $\mathcal{FL}_0$, which comprises the concept constructors conjunction ($\sqcap$) and value restriction ($\forall r.C$). The main reason for this focussing was that, when clarifying the logical status of property arcs in semantic networks and slots in frames (which are the ancestors of modern DLs), the decision was taken that arcs/slots should be read as value restrictions rather than existential restrictions ($\exists r.C$).

Unfortunately, as soon as TBoxes were taken into consideration, tractability turned out to be unattainable in $\mathcal{FL}_0$: even classifying the simplest form of TBoxes that admit only acyclic concept definitions was shown to be coNP-hard [Neb90]. If the most general form of TBoxes is admitted, which consists of general concept inclusion axioms (GCIs) as introduced in Section 2, then classification in $\mathcal{FL}_0$ even becomes ExpTime-complete [BBL05].

For these reasons, and also because of the need for expressive DLs in applications, from the mid 1990s on, the DL community had mainly given up on the quest of finding tractable DLs. Instead, it investigated more and more expressive DLs, for which reasoning is worst-case intractable. Recently, the choice of value restrictions as a sine qua non of DLs has been reconsidered. On the one hand, it was shown that the DL $\mathcal{EL}$, which allows for conjunction and existential restrictions, has better algorithmic properties than $\mathcal{FL}_0$. Classification of both acyclic and cyclic $\mathcal{EL}$ TBoxes is tractable [Baa03c], and this remains so even if general TBoxes with GCIs are admitted [Bra04]. A related line of research has identified a family of DLs (the DL-Lite family) that is closely related to $\mathcal{EL}$, but orthogonal in expressive power [CDGL$^+$05a, CGL$^+$06]. Just like $\mathcal{EL}$, many variants DL-Lite admit tractable classification in the presence of GCIs. More details are given in Section 14. On the other hand, there are applications where value restrictions are not needed, and where the expressive power of $\mathcal{EL}$ or small extensions thereof appear to be sufficient. In fact, the Systematized Nomenclature of Medicine (SNOMED), employs $\mathcal{EL}$ with an acyclic TBox [Spa00]. The Gene Ontology (GO) [Gen00] can be seen as an acyclic $\mathcal{EL}$ TBox with one transitive role. Finally, large parts of the Galen Medical Knowledge Base (GALEN) can also be expressed in $\mathcal{EL}$ with GCIs, role hierarchy, and transitive roles [RH97].

The tractability results for $\mathcal{EL}$ together with the bio-medical applications mentioned above have motivated our research on extensions of $\mathcal{EL}$: the leitmotif for this research was to extend $\mathcal{EL}$ as far as possible by adding standard DL constructors available in ontology languages like OWL, while still retaining polynomial-time reasoning in the presence of GCIs. Our first results in this direction have been published in [BBL05]. As part of the TONES project, we have considerably extended and refined our results about the $\mathcal{EL}$ family of DLs. As a result, the tractability border for classification in these DLs is now very well understood. In particular, we have identified two dialects called $\mathcal{EL}^{++}_{\text{rr}}$ and $\mathcal{EL}^{++}_{\text{ri}}$ for which classification is tractable and whose expressive power is well-suited for being used in life science ontologies. In this section, we present some selected results from this line of research. For the full picture, we refer to the paper [BBL07].

## 4.2 Description Logics

We introduce the description logic $\mathcal{EL}^{++}$ and two of its variations called $\mathcal{EL}^{++}_{\text{rr}}$ and $\mathcal{EL}^{++}_{\text{ri}}$. Concept descriptions of $\mathcal{EL}^{++}$ are formed using the constructors shown in the upper part of Table 1. The concrete domain constructor provides an interface to so-called concrete domains, which permits reference to, e.g., strings and integers. Formally, a *concrete domain* $\mathcal{D}$ is a pair $(\Delta^{\mathcal{D}}, \mathcal{P}^{\mathcal{D}})$ with $\Delta^{\mathcal{D}}$ a set and $\mathcal{P}^{\mathcal{D}}$ a set of *predicate names*. Each $p \in \mathcal{P}$ is associated with an arity $n > 0$ and an extension $p^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$. To provide a link between the DL and the concrete domain, we introduce a set of *feature names* $\mathsf{N_F}$. In Table 1, $p$ denotes a predicate of some concrete domain $\mathcal{D}$ and $f_1, \ldots, f_k$ are feature names. The DL $\mathcal{EL}^{++}$ may be equipped with a number of concrete domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ such that $\Delta^{\mathcal{D}_i} \cap \Delta^{\mathcal{D}_j} = \emptyset$ for $1 \leq i < j \leq n$. If we want to stress the use of particular concrete domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$, we write $\mathcal{EL}^{++}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ instead of $\mathcal{EL}^{++}$.

The semantics of $\mathcal{EL}^{++}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$-concept descriptions is defined in terms of an *interpretation* as introduced in Section 2, with the addition that each feature name $f \in \mathsf{N_F}$

| Name | Syntax | Semantics |
|---|---|---|
| top | $\top$ | $\Delta^{\mathcal{I}}$ |
| bottom | $\bot$ | $\emptyset$ |
| nominal | $\{a\}$ | $\{a^{\mathcal{I}}\}$ |
| conjunction | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| existential restriction | $\exists r.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| concrete domain | $p(f_1,\ldots,f_k)$ for $p \in \mathcal{P}^{\mathcal{D}_j}$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists y_1,\ldots,y_k \in \Delta^{\mathcal{D}_j} :$ $f_i^{\mathcal{I}}(x) = y_i$ for $1 \le i \le k \ \wedge (y_1,\ldots,y_k) \in p^{\mathcal{D}_j}\}$ |
| GCI | $C \sqsubseteq D$ | $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ |
| RI | $C\colon r_1 \circ \cdots \circ r_k \sqsubseteq r$ | $(C^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \cap (r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}}) \subseteq r^{\mathcal{I}}$ |
| domain restriction | $\mathsf{dom}(r) \sqsubseteq C$ | $r^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| range restriction | $\mathsf{ran}(r) \sqsubseteq C$ | $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}$ |
| 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 1: Syntax and semantics of $\mathcal{EL}^{++}$.

to a partial function $f^{\mathcal{I}}$ from $\Delta^{\mathcal{I}}$ to $\bigcup_{1 \le i \le n} \Delta^{\mathcal{D}_i}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is inductively defined as shown in the third column of Table 1.

An $\mathcal{EL}^{++}$ knowledge base (which corresponds to an ontology with an assertional part) comprises two sets, the constraint box (CBox) and the assertional box (ABox). While the CBox contains intensional knowledge defining the main notions relevant to the domain of discourse, the ABox contains extensional knowledge about actual named individuals of the domain. An $\mathcal{EL}^{++}$ *CBox* is a finite set of *constraints*. Such constraints are *general concept inclusions (GCIs)*, *role inclusions (RIs)*, *domain restrictions* and *range restrictions*, whose syntax can be found in Table 1. In role inclusions, we admit the case $k = 0$ and write the resulting inclusion as $\varepsilon \sqsubseteq r$. Note that a finite set of GCIs would commonly be called a *general TBox*. We use the term CBox due to the presence of RIs and domain and range restrictions. An interpretation $\mathcal{I}$ is a *model* of a CBox $\mathcal{C}$ if, for each constraint in $\mathcal{C}$, the conditions given in the third column of Table 1 are satisfied. In the definition of the semantics of RIs, the symbol "$\circ$" denotes composition of binary relations.

Apart from unrestricted $\mathcal{EL}^{++}$ CBoxes as defined above, two subsets of restricted CBoxes are of particular interest for us. We call every $\mathcal{EL}^{++}$ CBox $\mathcal{C}$ an $\mathcal{EL}^{++}_{\text{rr}}$ *CBox* iff all role inclusions in $\mathcal{C}$ are of the form $C\colon \varepsilon \sqsubseteq r$ (expressing reflexivity for $C$), $C\colon r \sqsubseteq s$ (role hierarchy for $C$), or $C\colon r \circ r \sqsubseteq r$ (transitivity for $C$). An $\mathcal{EL}^{++}$ CBox $\mathcal{C}$ is called $\mathcal{EL}^{++}_{\text{ri}}$ *CBox* iff $\mathcal{C}$ contains no range restrictions. In other words, $\mathcal{EL}^{++}_{\text{rr}}$ CBoxes admit arbitrary range restrictions but limited role inclusions while $\mathcal{EL}^{++}_{\text{ri}}$ CBoxes admit arbitrary role

inclusions but no range restrictions.

It remains to define the second component of our knowledge bases. An $\mathcal{EL}^{++}$ *ABox* is a finite set of *concept assertions* and *role assertions*, whose syntax can also be found in Table 1. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ if, for each concept assertion and role assertion in $\mathcal{A}$, the conditions given in the third column of Table 1 are satisfied.

In the remainder of the present paper, we will concentrate on subsumption as the basic reasoning task. This is justified by the facts that, first, all of the standard reasoning tasks "concept satisfiability", "concept subsumption", "ABox consistency", and the "instance problem" can be mutually polynomially reduced to one another, and second, subsumption is the most "traditional" reasoning service in description logics. We show mutual reducibility by reducing all (other) reasoning tasks to subsumption, and vice versa:

- Satisfiability to (non-)subsumption: a concept $C$ is satisfiable w.r.t. a CBox $\mathcal{C}$ iff $C \not\sqsubseteq_{\mathcal{C}} \bot$.
- Instance problem to subsumption. We convert an ABox $\mathcal{A}$ into a concept $C_{\mathcal{A}}$ as follows:
$$C_{\mathcal{A}} := \bigsqcap_{C(a)\in\mathcal{A}} \exists u.(\{a\} \sqcap C) \quad \sqcap \quad \bigsqcap_{r(a,b)\in\mathcal{A}} \exists u.(\{a\} \sqcap \exists r.\{b\})$$
where $u$ is a new role name not used in $\mathcal{A}$. Then, an individual $a$ is an instance of a concept $C$ in an ABox $\mathcal{A}$ w.r.t. a CBox $\mathcal{C}$ iff $\{a\} \sqcap C_{\mathcal{A}} \sqsubseteq_{\mathcal{C}} C$.
- Consistency to subsumption: $\mathcal{A}$ is consistent w.r.t. $\mathcal{C}$ iff $C_{\mathcal{A}} \not\sqsubseteq \bot$.
- Subsumption to satisfiability: $C \sqsubseteq_{\mathcal{C}} D$ iff $C \sqcap \{a\}$ is unsatisfiable w.r.t. the CBox $\mathcal{C} \cup \{D \sqcap \{a\} \sqsubseteq \bot\}$, where $a$ is an individual name not occurring in $C$, $D$, and $\mathcal{C}$.
- Subsumption to the instance problem: $C \sqsubseteq_{\mathcal{C}} D$ iff $a$ is an instance of $D$ in the ABox $\{C(a)\}$ w.r.t. $\mathcal{C}$.
- Subsumption to consistency: $C \sqsubseteq_{\mathcal{C}} D$ iff the ABox $\{C(a)\}$ is inconsistent w.r.t. the TBox $\mathcal{C} \cup \{D \sqcap \{a\} \sqsubseteq \bot\}$.

Three remarks regarding the expressivity of $\mathcal{EL}^{++}_{\mathrm{rr}}$ and $\mathcal{EL}^{++}_{\mathrm{ri}}$ are in order. First, our RIs generalize four means of expressivity important in ontology applications: *role hierarchies* $r \sqsubseteq s$, which can be expressed as $\top : r \sqsubseteq s$; *transitive roles*, which can be expressed by writing $\top : r \circ r \sqsubseteq r$; *reflexive roles* via $\top : \varepsilon \sqsubseteq r$; and so-called *right-identity rules* $r \circ s \sqsubseteq s$, expressible as $\top : r \circ s \sqsubseteq s$, which are important in medical applications [Spa00, HS03b]. Second, the bottom concept in combination with GCIs can be used to express *disjointness* of complex concept descriptions: $C \sqcap D \sqsubseteq \bot$ says that $C, D$ are disjoint. Finally, the *unique name assumption* for individual names can be enforced by writing $\{a\} \sqcap \{b\} \sqsubseteq \bot$ for all relevant individual names $a$ and $b$.

## 4.3 Deciding Subsumption in $\mathcal{EL}^{++}_{\mathrm{rr}}$ and $\mathcal{EL}^{++}_{\mathrm{ri}}$ extended by concrete domains

We develop a polynomial time algorithm for classification in $\mathcal{EL}^{++}_{\mathrm{rr}}$ and $\mathcal{EL}^{++}_{\mathrm{ri}}$, proceeding as follows. First, we develop an appropriate normal form for $\mathcal{EL}^{++}$ CBoxes; second, we

show how to eliminate domain and range restrictions from $\mathcal{EL}^{++}_{\text{rr}}$ CBoxes; and third, we show how to classify the resulting CBoxes, i.e., how to compute the subsumption relationships between all concept names in such a TBox.

### 4.3.1 A Normal Form for CBoxes

Given a CBox $\mathcal{C}$ formulated in $\mathcal{EL}^{++}$, we use $\mathsf{BC}_\mathcal{C}$ to denote the set of *basic concept descriptions for* $\mathcal{C}$ , i.e., the smallest set of concept descriptions that contains

- the top concept $\top$;
- all concept names used in $\mathcal{C}$;
- all (sub)concepts of the form $\{a\}$ or $p(f_1, \ldots, f_k)$ appearing in $\mathcal{C}$.

Now, a normal form for CBoxes can be defined as follows.

**Definition 4.1** [Normal Form for CBoxes] An $\mathcal{EL}^{++}$-CBox $\mathcal{C}$ is in *normal form* if

1. all concept inclusions have one of the following forms, where $C_1, C_2 \in \mathsf{BC}_\mathcal{C}$ and $D \in \mathsf{BC}_\mathcal{C} \cup \{\bot\}$:

$$\begin{aligned} C_1 &\sqsubseteq D \\ C_1 \sqcap C_2 &\sqsubseteq D \\ C_1 &\sqsubseteq \exists r.C_2 \\ \exists r.C_1 &\sqsubseteq D \end{aligned}$$

2. for all role inclusions $C\colon r_1 \circ \cdots \circ r_k \sqsubseteq r \in \mathcal{C}$, we have $C \in \mathsf{BC}_\mathcal{C}$ and $k \leq 2$;
3. there are no domain restrictions and all range restrictions are of the form $\mathsf{ran}(r) \sqsubseteq A$, where $A$ is a concept name.

By introducing new concept and role names, any CBox $\mathcal{C}$ can be turned into a normalized CBox $\mathcal{C}'$ such that every model of $\mathcal{C}'$ is also a model of $\mathcal{C}$, and every model of $\mathcal{C}$ can be extended to a model of $\mathcal{C}'$ by appropriate choice of the interpretations of the additional concept and role names.

This transformation can actually be done in linear time, yielding a normalized CBox $\mathcal{C}'$ whose size is linear in the size of $\mathcal{C}$. More precisely, this is achieved using the translation rules shown in Figure 1 in two phases:

1. exhaustively apply Rules **NF0** to **NF6**;
2. exhaustively apply Rules **NF7** to **NF9**.

Here "rule application" means that the concept inclusion on the left-hand side is replaced with the set of concept inclusions on the right-hand side, where Rule **NF4** is applied modulo commutativity of conjunction.

| | | | |
|---|---|---|---|
| **NF0** | $\hat{C}\colon r_1 \circ \cdots \circ r_k \sqsubseteq s$ | $\longrightarrow$ | $\{A \sqsubseteq C,\, C \sqsubseteq A,\, A\colon r_1 \circ \cdots \circ r_k \sqsubseteq s\}$ |
| **NF1** | $C\colon r_1 \circ \cdots \circ r_k \sqsubseteq s$ | $\longrightarrow$ | $\{C\colon r_1 \circ \cdots \circ r_{k-1} \sqsubseteq u,\, C\colon u \circ r_k \sqsubseteq s\}$ |
| **NF2** | $\mathsf{ran}(r) \sqsubseteq \hat{C}$ | $\longrightarrow$ | $\{\mathsf{ran}(r) \sqsubseteq A, A \sqsubseteq \hat{C}\}$ |
| **NF3** | $\mathsf{dom}(r) \sqsubseteq C$ | $\longrightarrow$ | $\{\exists r.\top \sqsubseteq C\}$ |
| **NF4** | $C \sqcap \hat{D} \sqsubseteq E$ | $\longrightarrow$ | $\{\ \hat{D} \sqsubseteq A, C \sqcap A \sqsubseteq E\ \}$ |
| **NF5** | $\exists r.\hat{C} \sqsubseteq D$ | $\longrightarrow$ | $\{\ \hat{C} \sqsubseteq A, \exists r.A \sqsubseteq D\ \}$ |
| **NF6** | $\bot \sqsubseteq D$ | $\longrightarrow$ | $\emptyset$ |
| **NF7** | $\hat{C} \sqsubseteq \hat{D}$ | $\longrightarrow$ | $\{\ \hat{C} \sqsubseteq A, A \sqsubseteq \hat{D}\ \}$ |
| **NF8** | $B \sqsubseteq \exists r.\hat{C}$ | $\longrightarrow$ | $\{\ B \sqsubseteq \exists r.A, A \sqsubseteq \hat{C}\ \}$ |
| **NF9** | $B \sqsubseteq C \sqcap D$ | $\longrightarrow$ | $\{\ B \sqsubseteq C, B \sqsubseteq D\ \}$ |

where $\hat{C}, \hat{D} \notin \mathsf{BC}_{\mathcal{C}}$, $u$ denotes a new role name, and $A$ a new concept name.

Figure 1: Normalization Rules

**Lemma 4.2** *Subsumption w.r.t. CBoxes in $\mathcal{EL}^{++}$ can be reduced in linear time to subsumption w.r.t. normalized CBoxes in $\mathcal{EL}^{++}$.*

Note that if all rules are applied together in one phase we obtain a quadratic blowup in the worst case due to the duplication of the concept $B$ by Rule **NF9**. Note also that, while domain restrictions are easily eliminated by means of Rule **NF3**, range restrictions remain unchanged by the above normalization procedure. In the following section, We therefore show how to remove range restrictions from $\mathcal{EL}^{++}_{\mathrm{rr}}$ CBoxes before presenting our subsumption algorithm in Section 4.3.3.

### 4.3.2 Eliminating Range Restrictions

Our aim is to devise a general strategy by which to remove range restrictions from any $\mathcal{EL}^{++}_{\mathrm{rr}}$ CBoxe $\mathcal{C}$ normalized in the sense of Definition 4.1. In order to simplify the presentation of our strategy, we begin by introducing some notation. For two roles $s$ and $r$, we write $s \sqsubseteq^*_{\mathcal{C}} r$ if there are roles $r_0, \ldots, r_n$, $n \geq 0$, such that $s = r_0$, $r = r_{n-1}$, and $r_i \sqsubseteq r_{i+1} \in \mathcal{C}$ for all $i < n$. Also, we write $r \in \mathsf{refl}_{\mathcal{C}}$ if there is a role $s$ such that $\varepsilon \sqsubseteq s$ and $s \sqsubseteq^*_{\mathcal{C}} r$.

To substitute a range restriction $\mathsf{ran}(r) \sqsubseteq A$ in $\mathcal{C}$, we proceed as follows:

1. exchange every GCI $C \sqsubseteq \exists s.D$ such that $s \sqsubseteq^* r$ with the three GCIs $C \sqsubseteq \exists s.B$, $B \sqsubseteq A$, and $B \sqsubseteq D$, where $B$ is a fresh concept name.

2. if $r \in \mathsf{refl}_{\mathcal{C}}$ then add the GCI $\top \sqsubseteq A$.

Let $\mathcal{C}'$ be the resulting CBox. Then we have the following.

**Lemma 4.3** *For all concept names $A, B$ occurring in $\mathcal{C}$, $A \sqsubseteq_{\mathcal{C}} B$ iff $A \sqsubseteq_{\mathcal{C}'} B$.*

Observe that eliminating range restrictions induces only a linear blowup in the size of the original CBox. Moreover, every normalized $\mathcal{EL}^{++}_{\text{rr}}$ CBox is by definition also a normalized $\mathcal{EL}^{++}_{\text{ri}}$ CBox. In the following, we may therefore assume w.l.o.g. to deal solely with normalized $\mathcal{EL}^{++}_{\text{ri}}$ CBoxes.

### 4.3.3 The Algorithm

We now develop a polynomial-time algorithm for deciding subsumption w.r.t. normalized $\mathcal{EL}^{++}_{\text{rr}}$ or $\mathcal{EL}^{++}_{\text{ri}}$ CBoxes. Note that every normalized $\mathcal{EL}^{++}_{\text{rr}}$ can also be viewed as an $\mathcal{EL}^{++}_{\text{ri}}$ CBox. Hence, we devise an algorithm for normalized $\mathcal{EL}^{++}_{\text{ri}}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ CBoxes. We can restrict our attention to subsumption between concept *names*. In fact, $C \sqsubseteq_{\mathcal{C}} D$ iff $A \sqsubseteq_{\mathcal{C}'} B$, where $\mathcal{C}' = \mathcal{C} \cup \{A \sqsubseteq C, D \sqsubseteq B\}$ with $A$ and $B$ new concept names. Our subsumption algorithm not only computes subsumption between two given concept names w.r.t. the normalized input CBox $\mathcal{C}$; it rather *classifies* $\mathcal{C}$, i.e., it simultaneously computes the subsumption relationships between *all* pairs of concept names occurring in $\mathcal{C}$.

Now, let $\mathcal{C}$ be an $\mathcal{EL}^{++}_{\text{ri}}$ CBox in normal form that is to be classified. We use $\mathsf{R}_{\mathcal{C}}$ to denote the set of all role names used in $\mathcal{C}$. The algorithm computes

- a mapping $S$ from $\mathsf{BC}_{\mathcal{C}}$ to a subset of $\mathsf{BC}_{\mathcal{C}} \cup \{\bot\}$, and
- a mapping $R$ from $\mathsf{R}_{\mathcal{C}}$ to a binary relation on $\mathsf{BC}_{\mathcal{C}}$.

The intuition is that these mappings make implicit subsumption relationships explicit in the following sense:

(I1) $D \in S(C)$ implies that $C \sqsubseteq_{\mathcal{C}} D$,

(I2) $(C, D) \in R(r)$ implies that $C \sqsubseteq_{\mathcal{C}} \exists r.D$.

In the algorithm, these mappings are initialized as follows:

- $S(C) := \{C, \top\}$ for each $C \in \mathsf{BC}_{\mathcal{C}}$,
- $R(r) := \emptyset$ for each $r \in \mathsf{R}_{\mathcal{C}}$.

Then the sets $S(C)$ and $R(r)$ are extended by applying the completion rules shown in Table 2 until no more rule applies.

Some of the rules use abbreviations that still need to be introduced. First, **CR6** uses the relation $\rightsquigarrow \subseteq \mathsf{BC}_{\mathcal{C}} \times \mathsf{BC}_{\mathcal{C}}$, which is defined as follows: $C \rightsquigarrow D$ iff there are $C_1, \ldots, C_k \in \mathsf{BC}_{\mathcal{C}}$ such that

- $C_1 = C$ or $C_1 = \{b\}$ for some individual name $b$,
- $(C_j, C_{j+1}) \in R(r_j)$ for some $r_j \in \mathsf{R}_{\mathcal{C}}$ $(1 \leq j < k)$,
- $C_k = D$.

| | |
|---|---|
| **CR1** | If $C' \in S(C)$, $C' \sqsubseteq D \in \mathcal{C}$, and $D \notin S(C)$<br>then $S(C) := S(C) \cup \{D\}$ |
| **CR2** | If $C_1, C_2 \in S(C)$, $C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{C}$, and $D \notin S(C)$<br>then $S(C) := S(C) \cup \{D\}$ |
| **CR3** | If $C' \in S(C)$, $C' \sqsubseteq \exists r.D \in \mathcal{C}$, and $(C, D) \notin R(r)$<br>then $R(r) := R(r) \cup \{(C, D)\}$ |
| **CR4** | If $(C, D) \in R(r)$, $D' \in S(D)$, $\exists r.D' \sqsubseteq E \in \mathcal{C}$, and $E \notin S(C)$<br>then $S(C) := S(C) \cup \{E\}$ |
| **CR5** | If $(C, D) \in R(r)$, $\bot \in S(D)$, and $\bot \notin S(C)$,<br>then $S(C) := S(C) \cup \{\bot\}$ |
| **CR6** | If $\{a\} \in S(C) \cap S(D)$, $C \rightsquigarrow D$, and $S(D) \not\subseteq S(C)$<br>then $S(C) := S(C) \cup S(D)$ |
| **CR7** | If $\mathsf{con}_j(S(C))$ is unsatisfiable in $\mathcal{D}_j$ and $\bot \notin S(C)$,<br>then $S(C) := S(C) \cup \{\bot\}$ |
| **CR8** | If $\mathsf{con}_j(S(C))$ implies $p(f_1, \ldots, f_k) \in \mathsf{BC}_{\mathcal{C}}$ in $\mathcal{D}_j$, and $p(f_1, \ldots, f_k) \notin S(C)$,<br>then $S(C) := S(C) \cup \{p(f_1, \ldots, f_k)\}$ |
| **CR9** | If $p(f_1, \ldots, f_k), p'(f'_1, \ldots, f'_{k'}) \in S(C)$, $p \in \mathcal{P}^{\mathcal{D}_j}$<br>$p' \in \mathcal{P}^{\mathcal{D}_\ell}$, $j \neq \ell$, $f_s = f'_t$ for some $s, t$, and $\bot \notin S(C)$<br>then $S(C) := S(C) \cup \{\bot\}$ |
| **CR10** | If $C \in \mathsf{BC}_{\mathcal{C}}$, $D \in S(C)$, $D\colon \varepsilon \sqsubseteq r \in \mathcal{C}$, and $(C, C) \notin R(s)$<br>then $R(r) := R(r) \cup \{(C, C)\}$ |
| **CR11** | If $(C, D) \in R(r)$, $E \in S(C)$, $E\colon r \sqsubseteq s \in \mathcal{C}$, and $(C, D) \notin R(s)$<br>then $R(s) := R(s) \cup \{(C, D)\}$ |
| **CR12** | If $(C, D) \in R(r_1)$, $(D, E) \in R(r_2)$, $F \in S(C)$,<br>$F\colon r_1 \circ r_2 \sqsubseteq r_3 \in \mathcal{C}$, and $(C, E) \notin R(r_3)$<br>then $R(r_3) := R(r_3) \cup \{(C, E)\}$ |

Table 2: Completion Rules

Second, rules **CR7** and **CR8** use the notion $\mathsf{con}_j(S_i(C))$, and satisfiability and implication in a concrete domain. If $p$ is a predicate of the concrete domain $\mathcal{D}_j$, then the $\mathcal{EL}^{++}$-concept description $p(f_1, \ldots, f_n)$ can be viewed as an atomic first-order formula with variables $f_1, \ldots, f_n$. Thus, it makes sense to consider Boolean combinations of such atomic formulae, and to talk about whether such a formula is satisfiable in (the first-order interpretation) $\mathcal{D}_j$, or whether in $\mathcal{D}_j$ one such formula implies another one. For a set $\Gamma$ of $\mathcal{EL}^{++}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$-concept descriptions and $1 \leq j \leq n$, we define

$$\mathsf{con}_j(\Gamma) := \bigwedge_{p(f_1,\ldots,f_k)\in\Gamma \text{ with } p\in\mathcal{P}^{\mathcal{D}_j}} p(f_1, \ldots, f_k).$$

For the rules **CR7** and **CR8** to be executable in polynomial time, satisfiability and implication in the concrete domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ must be decidable in polynominal time. However, for our algorithm to be complete, we must impose an additional condition on the concrete domains.

**Definition 4.4** The concrete domain $\mathcal{D}$ is *p-admissible* if

1. satisfiability and implication in $\mathcal{D}$ are decidable in polynominal time;
2. $\mathcal{D}$ is *convex*: if a conjunction of atoms of the form $p(f_1, \ldots, f_k)$ implies a disjunction of such atoms, then it also implies one of its disjuncts.

We investigate the property of p-admissibility in more detail in Section 4.4, where we also exhibit some useful concrete domains that are p-admissible.

The next lemma shows how all subsumption relationships between concept names occurring in $\mathcal{C}$ can be determined once the completion algorithm has terminated.

**Lemma 4.5** *Let $S$ be the mapping obtained after the application of the rules of Table 2 to the normalized CBox $\mathcal{C}$ has terminated, and let $A, B$ be concept names occurring in $\mathcal{C}$. Then $A \sqsubseteq_{\mathcal{C}} B$ iff one of the following two conditions holds:*

- $S(A) \cap \{B, \bot\} \neq \emptyset$,
- *there is an $\{a\} \in \mathsf{BC}_{\mathcal{C}}$ such that $\bot \in S(\{a\})$.*

Lemma 4.5 is proved in [BBL07], where it is also shown that the algorithm terminates after at most polynomially many rule applications. Here, we briefly discuss soundness of the algorithm on an intuitive level. Soundness immediately follows from the fact that (I1) and (I2) are satisfied for the initial definition of $S, R$, and that application of the rules preserves (I1) and (I2). This is trivially seen for most of the rules. However, it is worthwhile to consider **CR6** in more detail. If $\{a\} \in S(C) \cap S(D)$, then $C, D \sqsubseteq_{\mathcal{C}} \{a\}$. Now, $C \rightsquigarrow D$ implies that $C \sqsubseteq_{\mathcal{C}} \exists r_1. \cdots \exists r_{k-1}.D$ or $\{b\} \sqsubseteq_{\mathcal{C}} \exists r_1. \cdots \exists r_{k-1}.D$ for some individual name $b$. In the second case, this implies that $D$ cannot be empty in any model of $\mathcal{C}$, and in the first case it implies that $D$ is non-empty in any model of $\mathcal{C}$ for which $C$ is non-empty. Together with $C, D \sqsubseteq_{\mathcal{C}} \{a\}$, this implies that $C \sqsubseteq_{\mathcal{C}} D$, which shows that the rule **CR6** is sound since it preserves (I1). When dropping the requirement $C \rightsquigarrow D$ from this rule, (I1) is no longer preserved.

Finally, we obtain the following result.

**Theorem 4.6** *Satisfiability, subsumption, ABox consistency, and the instance problem in $\mathcal{EL}^{++}_{\mathrm{rr}}$ and $\mathcal{EL}^{++}_{\mathrm{ri}}$ can be decided in polynomial time.*

## 4.4 P-admissible and Non-admissible Concrete Domains

In the previous sections we have dealt with p-admissible concrete domains without actually specifying one. In order to obtain concrete DLs of the form $\mathcal{EL}^{++}_{\mathrm{rr}}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ and $\mathcal{EL}^{++}_{\mathrm{ri}}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ for $n > 0$ to which Theorem 4.6 applies, actual p-admissible concrete domains are needed. In the following, we introduce two p-admissible concrete domains and show that the property of p-admissibility is lost in small extensions of them. To simplify notation, we call every finite conjunction of atomic formulae $p(f_1, \ldots, f_k)$ from a concrete domain $\mathcal{D}$ a *$\mathcal{D}$-conjunction*.

The concrete domain $\mathsf{Q} = (\mathbb{Q}, \mathcal{P}^{\mathsf{Q}})$ has as its domain the set $\mathbb{Q}$ of rational numbers, and its set of predicates $\mathcal{P}^{\mathsf{Q}}$ consists of the following predicates:

- a unary predicate $\top_{\mathsf{Q}}$ with $(\top_{\mathsf{Q}})^{\mathsf{Q}} = \mathbb{Q}$;
- unary predicates $=_q$ and $>_q$ for each $q \in \mathbb{Q}$;
- a binary predicate $=$;
- a binary predicate $+_q$, for each $q \in \mathbb{Q}$, with $(+_q)^{\mathsf{Q}} = \{(q', q'') \in \mathbb{Q}^2 \mid q' + q = q''\}$.

The concrete domain $\mathsf{S}$ is defined as $(\Sigma^*, \mathcal{P}^{\mathsf{S}})$, where $\Sigma$ is the ISO 8859-1 (Latin-1) character set and $\mathcal{P}^{\mathsf{S}}$ consistes of the following predicates:

- a unary predicate $\top_{\mathsf{S}}$ with $(\top_{\mathsf{S}})^{\mathsf{S}} = \Sigma^*$;
- a unary predicate $=_w$, for each $w \in \Sigma^*$;
- a binary predicate $=$;
- a binary predicate $\mathsf{conc}_w$, for each $w \in \Sigma^*$, with $\mathsf{conc}_w^{\mathsf{Q}} = \{(w', w'') \mid w'' = w'w\}$.

Both $\mathsf{Q}$ and $\mathsf{S}$ are interesting concrete domains since they allow us to refer to concrete numbers and strings in concepts, and use the properties of the concrete predicates when reasoning. However, the predicates available in these concrete domains are rather restricted. We now show that both $\mathsf{Q}$ and $\mathsf{S}$ are p-admissible.

**Proposition 4.7** *The concrete domains $\mathsf{Q}$ and $\mathsf{S}$ are p-admissible.*

Note that p-admissibility of concrete domains is easily broken. Consider e.g. the following examples:

- The concrete domain $\mathsf{Q}^{<_q, >_q}$ with domain $\mathbb{Q}$ that has the predicates $(>_q)_{q \in \mathbb{Q}}$ from $\mathsf{Q}$ and, additionally, unary predicates $(<_q)_{q \in \mathbb{Q}}$ with
$$(<_q)^{\mathsf{Q}^{<_q, >_q}} := \{q' \in \mathbb{Q} \mid q' < q\}.$$
Then the $\mathsf{Q}^{<_q, >_q}$-conjunction $c := >_0(f')$ does not imply any concept from $\Gamma := \{<_0(f), =_0(f), >_0(f)\}$, but every solution of $c$ satisfies some concept of $\Gamma$.

| Name | Syntax | Semantics |
|---|---|---|
| negation | $\neg C$ | $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| disjunction | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| value restriction | $\forall r.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ |
| sink restriction | $\forall r.\bot$ | $\{x \in \Delta^{\mathcal{I}} \mid \neg\exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}}\}$ |
| at-least restriction | $(\geqslant n\ r)$ | $\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$ |
| at-most restriction | $(\leqslant n\ r)$ | $\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$ |
| inverse roles | $\exists r^{-}.C$ | $\{x \mid \exists y \in \Delta^{\mathcal{I}} : (y,x) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| role negation | $\exists \neg r.C$ | $\{x \mid \exists y \in \Delta^{\mathcal{I}} : (y,x) \notin r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| role union | $\exists r \cup s.C$ | $\{x \mid \exists y \in \Delta^{\mathcal{I}} : (y,x) \in r^{\mathcal{I}} \cup s^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| transitive closure | $\exists r^{*}.C$ | $\{x \mid \exists y \in \Delta^{\mathcal{I}} : (y,x) \in (r^{\mathcal{I}})^{+} \wedge y \in C^{\mathcal{I}}\}$ |
| symmetric roles | — | subset $\mathsf{N}_{\mathsf{R}}^{\mathsf{sym}} \subseteq \mathsf{N}_{\mathsf{R}}$ such that for all $r \in \mathsf{N}_{\mathsf{R}}^{\mathsf{sym}}$, $r^{\mathcal{I}} = r^{\mathcal{I}} \subseteq \{(y,x) \mid (x,y) \in r^{\mathcal{I}}\}$ |

Table 3: The additional constructors.

- Any concrete domain $\mathsf{S}^*$ with domain $\Sigma^*$ for some finite alphabet $\Sigma$ and the unary predicates $\mathsf{pref}_s$ and $\mathsf{suff}_s$ for every $s \in \Sigma^*$ with

$$\begin{aligned} \mathsf{pref}_s^{\mathsf{S}^*} &:= \{s' \mid s \text{ is a prefix of } s'\} \\ \mathsf{suff}_s^{\mathsf{S}^*} &:= \{s' \mid s \text{ is a suffix of } s'\} \end{aligned}$$

  Assume $a \in \Sigma$. Then the $S^*$-conjunction $c := \mathsf{suff}_a(f)$ implies no formula from $\Gamma := \{\mathsf{pref}_\sigma(f) \mid \sigma \in \Sigma\}$, but every solution of $c$ satisfies some formula from $\Gamma$.

- Any concrete domain $\mathsf{S}^*$ with domain $\Sigma^*$ for some finite alphabet $\Sigma$, the unary predicates $\top_{\mathsf{S}^*}$ and $=_\varepsilon$ with the obvious semantics, and the unary predicates $\mathsf{pref}_s$, $s \in \Sigma^*$, as in the previous example. Then the $S^*$-conjunction $c := \top_{\mathsf{S}^*}(f)$ implies no concept from $\Gamma := \{=_\varepsilon(f)\} \cup \{\mathsf{pref}_\sigma(f) \mid \sigma \in \Sigma\}$, but every solution of $c$ satisfies some concept from $\Gamma$.

## 4.5 Lower Bounds

The purpose of this section is to justify our choice of constructors in the languages $\mathcal{EL}^{++}_{\mathrm{rr}}$ and $\mathcal{EL}^{++}_{\mathrm{ri}}$. We start by showing that subsumption w.r.t. CBoxes in full $\mathcal{EL}^{++}$ is undecidable. Then, we consider the sublanguage $\mathcal{EL}$ of $\mathcal{EL}^{++}$ and restrict the attention to general TBoxes, i.e., finite sets of GCIs. Recall that $\mathcal{EL}$is obtained from $\mathcal{EL}^{++}$ by dropping all concept constructors except conjunction, existential restriction, and top. We will show that the extension of $\mathcal{EL}$ with basically any typical DL constructor not present in $\mathcal{EL}^{++}$ results in subsumption w.r.t. general TBoxes becoming ExpTime-complete. In this deliverable, we present only the proofs for three selected cases and refer to [BBL07] for full details. A list of the constructors whose addtion to $\mathcal{EL}$ leads to ExpTime-completeness can be found in Table 3, along with the syntax and semantics. Here, $\#S$ denotes the cardinality of a set $S$ and $(r^{\mathcal{I}})^{+}$ denotes the transitive closure of the relation $r^{\mathcal{I}}$.

### 4.5.1 Undecidability of Full $\mathcal{EL}^{++}$

We show that subsumption w.r.t. CBoxes in $\mathcal{EL}^{++}$ is undecidable. Our proof applies to the fragment of $\mathcal{EL}^{++}$ that has conjunction and existential restriction as the only concept constructors, and GCIs, RIs, and range restrictions as the only constraints in CBoxes. The proof uses a reduction of the emptyness problem of the intersection of two context-free grammars [HU97, Theorem 8.10].

Recall that a context free grammar is a tuple $(\Sigma, N, P, S)$ with $\Sigma$ a finite alphabet of *terminal symbols*, $N$ a finite set of *non-terminal symbols*, $S \in N$ a *start symbol*, and $P \subseteq N \times (\Sigma \cup N)^*$ a finite set of *productions*. We denote the lanuage generated by a grammar $G$ with $L(G)$.

Let $G = (\Sigma, N, P, S)$ and $G' = (\Sigma, N', P', S')$ be two context-free grammars. W.l.o.g., we may assume that $N \cap N' = \emptyset$. We show how to translate $G$ and $G'$ into a CBox $\mathcal{C}$ and concept names $A$ and $B$ such that $L(G) \cap L(G') = \emptyset$ iff $A \not\sqsubseteq_{\mathcal{C}} B$. In the CBox $\mathcal{C}$, we use two role names $r_x$ and $r'_x$ for every $x \in \Sigma \cup N$ and three additional concept names $A'$, $X$ and $Y$. More precisely, $\mathcal{C}$ contains the following constraints:

1. the GCIs $A \sqsubseteq \bigsqcap_{a \in \Sigma} \exists r'_a.A'$ and $A' \sqsubseteq \bigsqcap_{a \in \Sigma} \exists r_a.A'$;
2. for every $v \vdash x_1 \cdots x_n \in P \cup P'$, the RIs $\top : r_{x_1} \circ \cdots \circ r_{x_n} \sqsubseteq r_v$ and $\top : r'_{x_1} \circ \cdots \circ r_{x_n} \sqsubseteq r'_v$;
3. the range restrictions $\mathsf{ran}(r'_S) \sqsubseteq X$ and $\mathsf{ran}(r'_{S'}) \sqsubseteq Y$;
4. the GCI $X \sqcap Y \sqsubseteq B$ and for each $a \in \Sigma$, the GCIs $\exists r_a.B \sqsubseteq B$ and $\exists r'_a.B \sqsubseteq B$.

This translation is as desired.

**Lemma 4.8** *$L(G) \cap L(G') = \emptyset$ iff $A \not\sqsubseteq_{\mathcal{C}} B$.*

We also mention a related undecidability result here. It shows that in $\mathcal{EL}^{++}_{ri}$, we cannot strength role inclusions to so-called role value maps without losing decidability. Formally, a *role-value-maps (RVMs)* is an inclusions $\top : r_1 \circ \cdots \circ r_k \sqsubseteq s_1 \circ \cdots \circ s_\ell$ whose right-hand side is a *composition* of role names. The semantics of RVMs is defined in analogy with the semantics of $\mathcal{EL}^{++}$'s role inclusions. By a result of Baader [Baa03c], subsumption in $\mathcal{EL}$is undecidable already if only RVMs, but no concept inclusions and no domain and range restrictions are admitted in CBoxes.

**Theorem 4.9 (Baader)** *Subsumption of $\mathcal{EL}$-concepts w.r.t. RVMs is undecidable.*

In the following, we walk through the constructors listed in Table 3 and, for each of them, prove that subsumption w.r.t. general TBoxes is not tractable.

### 4.5.2 Atomic negation

Let $\mathcal{EL}^{\neg}$ be the extension of $\mathcal{EL}$with negation, and let $\mathcal{EL}^{(\neg)}$ be obtained from $\mathcal{EL}^{\neg}$ by restricting the applicability of negation to concept names (*atomic* negation). Since $\mathcal{EL}^{\neg}$ is a notational variant of the DL $\mathcal{ALC}$, ExpTime-completeness of satisfiability and subsumption in $\mathcal{ALC}$ w.r.t. general TBoxes [Sch91] carries over to $\mathcal{EL}^{\neg}$. ExpTime-completeness

even carries over to $\mathcal{EL}^{(\neg)}$ since $\neg C$ with $C$ complex can be replaced with $\neg A$ for a new concept name $A$ if we add the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$.

**Theorem 4.10** *In $\mathcal{EL}^{(\neg)}$, satisfiability and subsumption w.r.t. general TBoxes is* ExpTime*-complete.*

For many other extensions of $\mathcal{EL}$presented in this section, satisfiability is trivial in the sense that every concept is satisfiable w.r.t. every TBox. In the following, we will only explicitly mention satisfiability if it is *not* trivial.

### 4.5.3 Disjunction

Let $\mathcal{ELU}$ be the extension of $\mathcal{EL}$with disjunction. Our aim is to show that subsumption in $\mathcal{ELU}$ w.r.t. general TBoxes is ExpTime-complete. The upper bound follows from $\mathcal{ELU}$ being a fragment of $\mathcal{ALC}$. For the lower bound, we reduce *satisfiability* of $\mathcal{EL}^{(\neg)}$-concepts w.r.t. general TBoxes to subsumption of $\mathcal{ELU}$-concepts. The former is ExpTime-hard by Theorem 4.10.

Let $A_0$ be an $\mathcal{EL}^{(\neg)}$ concept name and $\mathcal{T}$ a general $\mathcal{EL}^{(\neg)}$ TBox. To decide satisfiability of $A_0$ w.r.t. $\mathcal{T}$, take a new (i.e. distinct from $A_0$ and not occurring in $\mathcal{T}$) concept name $A'$ for each concept name $A$ occurring in $\mathcal{T}$. Also fix an additional new concept name $L$. Then the TBox $\mathcal{T}^*$ is obtained from $\mathcal{T}$ by first replacing each subconcept $\neg A$ with $A'$, and then adding the following GCIs:

- $\top \sqsubseteq A \sqcup A'$ and $A \sqcap A' \sqsubseteq L$ for each concept name $A$ occurring in $\mathcal{T}$;
- $\exists r.L \sqsubseteq L$ for every role $r$ occurring in $\mathcal{T}$.

Note that the concept inclusion $\exists r.L \sqsubseteq L$ is equivalent to $\neg L \sqsubseteq \forall r.\neg L$. It thus ensures that $L$ acts as the bottom concept in (connected) countermodels of the subsumption $A_0 \sqsubseteq_{\mathcal{T}^*} L$. Using this observation, it is not hard to verify that $C$ is satisfiable w.r.t. $\mathcal{T}$ if, and only if, $A_0 \not\sqsubseteq_{\mathcal{T}^*} L$.

**Theorem 4.11** *In $\mathcal{ELU}$, subsumption w.r.t. general TBoxes is* ExpTime*-complete.*

This theorem improves upon the result of Brandt that subsumption of $\mathcal{ELU}$ concepts w.r.t. general TBoxes is NP-hard [Bra04], and it improves upon the result of Hladik and Sattler that satisfiability of $\mathcal{ELU}$ concepts extended with functional roles and the bottom concept w.r.t. general TBoxes is ExpTime-hard [HS03a].

### 4.5.4 At-Least Restrictions

Let $\mathcal{EL}^{\geq 2}$ be the extension of $\mathcal{EL}$ with at-least restrictions of the form $(\geqslant 2\ r)$. Subsumption in $\mathcal{EL}^{\geq 2}$ w.r.t. general TBoxes is in ExpTime since $\mathcal{EL}^{\geq 2}$ is a fragment of $\mathcal{ALC}$ extended with number restrictions [GL94]. We establish a matching lower bound by reducing subsumption in $\mathcal{ELU}$ w.r.t. general TBoxes. Let $A_0$ and $B_0$ be concept names

and $\mathcal{T}$ a general $\mathcal{ELU}$ TBox. We assume that all concept inclusions in $\mathcal{T}$ have one of the following forms:

$$\begin{aligned} C &\sqsubseteq D \\ C_1 \sqcap C_2 &\sqsubseteq C \\ C &\sqsubseteq C_1 \sqcup C_2 \\ C &\sqsubseteq \exists r.D \\ \exists r.C &\sqsubseteq D \end{aligned}$$

where $C$, $D$, $C_1$, and $C_2$ are concept names or $\top$. It is easily verified that this assumption can be made without loss of generality since every general TBox can be converted into normal form using normalization rules similar to the one presented in Figure 1. Note in particular that $C_1 \sqcup C_2 \sqsubseteq C$ can be replaced by the two rules $C_1 \sqsubseteq C$ and $C_2 \sqsubseteq C$. To convert $\mathcal{T}$ into an $\mathcal{EL}^{\geq 2}$ CBox, we only need to rephrase concept implications of the form $C \sqsubseteq C_1 \sqcup C_2$. This is done as follows: introduce two new concept names $A$ and $B$ and a new role name $r$, and replace the mentioned implication with

$$\begin{aligned} C &\sqsubseteq \exists r.A \sqcap \exists r.B \\ C \sqcap \exists r.(A \sqcap B) &\sqsubseteq C_1 \\ C \sqcap (\geqslant 2\, r) &\sqsubseteq C_2 \end{aligned}$$

Call the resulting TBox $\mathcal{T}^*$. It is easily seen that $A_0 \sqsubseteq_{\mathcal{T}} B_0$ iff $A_0 \sqsubseteq_{\mathcal{T}^*} B_0$.

**Theorem 4.12** *In $\mathcal{EL}^{\geq 2}$, subsumption w.r.t. general TBoxes is* ExpTime*-complete.*

## 4.6 Related Work

Our results show that the quest for tractable DLs that are expressive enough to be useful in practice can be successful. Our DL formalisms $\mathcal{EL}^{++}_{\text{rr}}$ and $\mathcal{EL}^{++}_{\text{ri}}$ are tractable even w.r.t. GCIs, and offer many constructors that are important in ontology applications. This is in strong contrast to their counterpart with value restrictions: $\mathcal{FL}_0$ is tractable without TBoxes [BL84], co-NP-complete for acyclic TBoxes [Neb90], PSpace-complete for cyclic TBoxes [Baa96, KdN03], and ExpTime-complete for general TBoxes (as shown above, and, independently, in [Hof05]).

As has already been mentioned, there is a closely related line of research within the TONES project, namely that on the DL-Lite family of description logics, see [CDGL+05a, CGL+06] and Section 14. In a nutshell, the main difference between the $\mathcal{EL}^{++}$ and DL-Lite families of DLs is that, in the latter, the existential restriction $\exists r.C$ is only allowed in the form $\exists r.\top$. This then allows to add features such as inverse roles without losing tractability of reasoning w.r.t. GCIs; recall that, in contrast, $\mathcal{EL}^{++}$ with inverse roles and GCIs is ExpTime-complete.

As part of the TONES project, a reasoner for $\mathcal{EL}^{++}_{\text{rr}}$ and $\mathcal{EL}^{++}_{\text{ri}}$ CBoxes is currently being implemented. In its current state, the reasoner CEL [BLS06] supports $\mathcal{EL}^{+}$-CBoxes, where $\mathcal{EL}^{+}$ denotes the fragment of $\mathcal{EL}^{++}_{\text{ri}}$ without the bottom concept, nominals, and concrete domains. Both GCIs and RIs are fully supported in the TBox formalism. In order to achieve high efficiency, the algorithm presented in Section 4.3.3 needs to be significantly modified. This will be the subject of a future deliverable.

# 5 Reasoning in OWL 1.1 and $\mathcal{SROIQ}$

## 5.1 Introduction

Since the inception of the Semantic Web, the development of a language for modeling ontologies—conceptualizations of a domain shared by a community of users—was seen as a key task. The initial proposals focused on RDF and RDF Schema; however, these languages were soon found to be too limited in expressive power [HPSvH03]. The World Wide Web Consortium (W3C) therefore formed the Web Ontology Language working group, whose goal was to develop an expressive language suitable for application in the Semantic Web. The result of this endeavor was the OWL Web Ontology Language, which became a W3C recommendation in February, 2004. OWL is actually a family of three language variants (often called *species*) of increasing expressive power: OWL Lite, OWL DL, and OWL Full.

The standardization of OWL has sparked the development and/or adaption of a number of OWL-based reasoners[1] and ontology editors such as Protege and Swoop. Despite the success story surrounding OWL, the numerous contexts in which the language has been applied, OWL is still lacking some of the expressive power required for many applications.

In response to user comments and requests, the idea was born to address some of these needs via an incremental revision of OWL, called OWL 1.1. The initial goal was to exploit recent developments in DL research in order to address some of the expressivity limitations of the language.

The drawbacks of OWL 1.0 regarding expressivity have long been recognized in the DL community, and a significant amount of research has been devoted to finding possible solutions. The cumulative results of this work are embodied in the DL $\mathcal{SROIQ}$, which constitutes the logical underpinning for OWL 1.1.

Extending the ontology language, however, poses new challenges. In particular, existing reasoning algorithms must be extended to deal with the new constructs. This reasoning support is key for assisting the user in crucial tasks in ontology design. In particular, the availability of algorithms for ontology consistency, concept satisfiability and classification is fundamental for the *formulation of concept descriptions*, which are finally to constitute the ontology. Indeed, experience in ontology design shows that even users who are experts in both the ontology language and the domain to be formalized often find it difficult to construct adequate and comprehensive concept descriptions.

The most important reasoning services for assisting the user in the construction of both concept descriptions and axioms in the ontology are the following:

- *Inconsistent Ontology.* The reasoning backend takes an ontology as input and determines whether the whole ontology is consistent. Intuitively, an ontology is consistent if there exists at least a single possible state of the world that matches the concept descriptions contained in the ontology. An inconsistent ontology always indicates a serious modeling flaw, and may be the result of adding a new concept description that interacts with the existing ones in an unintended way. If an inconsistent ontol-

[1] See http://www.cs.man.ac.uk/∼sattler/reasoners.html for a list of reasoners.

ogy occurs not as the result of adding a new concept, it may be difficult to remove the inconsistency because many concept descriptions that interact in a subtle way may be involved in it.

- *Inconsistent Concept Descriptions.* The reasoning backend takes an ontology and a concept description as input and determines whether or not the description is inconsistent w.r.t. the ontology. In particular, the reasoner takes into account the impact of the surrounding ontology as described above. An inconsistent concept is a concept that cannot have any instances. Note that all concepts contained in an inconsistent ontology are inconsistent w.r.t. this ontology, but there may be inconsistent concepts in a consistent ontology. In this sense, inconsistent concepts are a less severe problem than an inconsistent ontology. Nevertheless, they usually indicate a modeling mistake that requires inspection by the ontology designer.

- *Position in Subsumption Hierarchy:* To properly understand the implicit consequences of a new concept description, it is usually advisable to determine the position of a new concept in the subsumption hierarchy. To achieve this, the reasoning backend takes the modified ontology and recomputes the subsumption hierarchy that can then be displayed to the user. If undesired subsumption relationships show up or desired ones are missing, the ontology designer should reexamine the new concept description to resolve the problem.

- *Detection of Equivalent Concepts:* Large ontologies are often designed by a group of collaborating designers rather than by a single individual. In such an environment, it frequently happens that different designers inadvertedly introduce the same concept into the ontology twice, using different names. This may happen e.g. when the designers work on different subareas of the application domain that are both related to the concept in question. A reasoning backend can take an ontology and a concept as input, and return all the concepts in the ontology that are equivalent to the given one. A detected equivalence is usually a sign for redundancy in the ontology or for a modeling mistake. It therefore requires inspection by an ontology designer.

In this Section, we develop algorithmic techniques for solving these reasoning problems in very expressive ontology languages. In particular, we develop algorithms for ontology consistency, concept satisfiability and concept subsumption in the description logic $\mathcal{SROIQ}$, which provides the logical underpinning for OWL 1.1. We will assume that reasoning is performed on a single, stand-alone, OWL 1.1 ontology $\mathcal{T}$ and thus we will be dealing with the simplest case in our framework for representing ontologies.

## 5.2 The Description Logic $\mathcal{SROIQ}$

In this section, we describe the logic $\mathcal{SROIQ}$, which is an extension of the $\mathcal{SHOIQ}$ description logic described in Section 2. The logic $\mathcal{SROIQ}$ extends $\mathcal{SHOIQ}$ with the following functionality:

1. *disjoint roles.* Most DLs can be said to be "unbalanced" since they allow to express disjointness on concepts but not on roles, despite the fact that role disjointness is

quite natural and can generate new subsumptions or inconsistencies in the presence of role hierarchies and number restrictions. E.g., the roles `sister` and `mother` should be declared as being disjoint.

2. *reflexive, irreflexive, and antisymmetric roles.* These features are of minor interest when considering only TBoxes not using nominals, yet they add some useful constraints if we also refer to individuals, either by using nominals or ABoxes, especially in the presence of number restrictions. E.g., the roles `knows`, `hasSibling`, and `properPartOf`, should be declared as, respectively, reflexive, irreflexive, and antisymmetric.

3. *negated role assertions.* Most Abox formalisms only allow for positive role assertions, whereas $\mathcal{SROIQ}$ also allows for statements like $(\texttt{John}, \texttt{Mary}) : \neg\texttt{likes}$. In the presence of complex role inclusions, negated role assertions can be quite useful and, like disjoint roles, they overcome a certain asymmetry in expressivity.

4. $\mathcal{SROIQ}$ provides complex role inclusion axioms of the form $R \circ S \sqsubseteq R$ and $S \circ R \sqsubseteq R$. For example, w.r.t. the axiom $\texttt{owns} \circ \texttt{hasPart} \sqsubseteq \texttt{owns}$, and the fact that each car contains an engine $\texttt{Car} \sqsubseteq \exists\texttt{hasPart}.\texttt{Engine}$, an owner of a car is also an owner of an engine, i.e., the following subsumption holds: $\exists\texttt{owns}.\texttt{Car} \sqsubseteq \exists\texttt{owns}.\texttt{Engine}$.

5. $\mathcal{SROIQ}$ provides the *universal role* $U$. Together with nominals (which are also provided by $\mathcal{SHOIQ}$), this role is a prominent feature of hybrid logics [BS95]. Nominals can be viewed as a powerful generalisation of *ABox individuals* [Sch94, HS01], and they occur naturally in ontologies, e.g., when describing a class such as $\mathsf{EUCountries}$ by enumerating its members.

6. Finally, $\mathcal{SROIQ}$ allows for concepts of the form $\exists R.\mathsf{Self}$ which can be used to express "local reflexivity" of a role $R$, e.g., to define the concept "narcist" as $\exists\texttt{likes}.\mathsf{Self}$.

$\mathcal{SROIQ}$ is designed to be of similar practicability as $\mathcal{SHOIQ}$. Even though the additional expressive means require certain adjustments, these adjustments do not add new sources of non-determinism and, subject to empirical verification, are believed to be "harmless" in the sense of not significantly degrading typical performance as compared with the $\mathcal{SHOIQ}$ algorithm.

**Definition 5.1** Let $\mathbf{C}$ be a set of *concept names* including a subset $\mathbf{N}$ of *nominals*, $\mathbf{R}$ a set of *role names* including the universal role $U$, and $\mathsf{N_N} = \{a, b, c \ldots\}$ a set of *individual names*. The set of *roles* is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$, where a role $R^-$ is called the *inverse role* of $R$.

As usual, an *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of a set $\Delta^\mathcal{I}$, called the *domain* of $\mathcal{I}$, and a *valuation* $\cdot^\mathcal{I}$ which associates, with each role name $R$, a binary relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, with the universal role $U$ the universal relation $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$, with each concept name $C$ a subset $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$, where $C^\mathcal{I}$ is a singleton set if $C \in \mathbf{N}$, and, with each individual name $a$, an element $a^\mathcal{I} \in \Delta^\mathcal{I}$. Inverse roles are interpreted as usual, i.e., for each role $R \in \mathbf{R}$, we have

$$(R^-)^\mathcal{I} = \{(y, x) \mid (x, y) \in R^\mathcal{I}\}.$$

Obviously, $(U^-)^{\mathcal{I}} = (U)^{\mathcal{I}}$. Note that, unlike in the cases of $\mathcal{SHIQ}$ or $\mathcal{SHOIQ}$, we did not introduce *transitive role names*. This is because, as will become apparent below, role box assertions can be used to force roles to be transitive.

To avoid considering roles such as $R^{--}$, we define a function $\mathsf{Inv}$ on roles such that $\mathsf{Inv}(R) = R^-$ if $R \in \mathbf{R}$ is a role name, and $\mathsf{Inv}(R) = S \in \mathbf{R}$ if $R = S^-$.

Since we will often work with finite strings of roles it is convenient to extend both $\cdot^{\mathcal{I}}$ and $\mathsf{Inv}(\cdot)$ to such strings: if $w = R_1 \dots R_n$ is a string of roles $R_i$ $(1 \leq i \leq n)$, we set $\mathsf{Inv}(w) = \mathsf{Inv}(R_n) \dots \mathsf{Inv}(R_1)$ and $w^{\mathcal{I}} = R_1^{\mathcal{I}} \circ \dots \circ R_n^{\mathcal{I}}$, where $\circ$ denotes composition of binary relations.

A *role box* $\mathcal{R}$ consists of two components. The first component is a *role hierarchy* $\mathcal{R}_h$ which consists of (generalised) *role inclusion axioms*. The second component is a set $\mathcal{R}_a$ of *role assertions* stating, for instance, that a role $R$ must be interpreted as an irreflexive relation.

We start with the definition of a (regular) role hierarchy whose definition involves a certain ordering on roles, called *regular*. A strict partial order $\prec$ on a set $A$ is an irreflexive and transitive relation on $A$. A strict partial order $\prec$ on the set of roles $R \cup \{R^- \mid R \in \mathbf{R}\}$ is called a *regular order* if $\prec$ satisfies, additionally, $S \prec R \iff S^- \prec R$, for all roles $R$ and $S$. Note, in particular, that the irreflexivity of $\prec$ ensures that neither $S^- \prec S$ nor $S \prec S^-$ hold.

**Definition 5.2** [(Regular) Role Inclusion Axioms]

Let $\prec$ be a regular order on roles. A *role inclusion axiom* (RIA for short) is an expression of the form $w \dot{\sqsubseteq} R$, where $w$ is a finite string of roles not including the universal role $U$, and $R \neq U$ is a role name. A *role hierarchy* $\mathcal{R}_h$ is a finite set of RIAs. An interpretation $\mathcal{I}$ *satisfies* a role inclusion axiom $w \dot{\sqsubseteq} R$, written $\mathcal{I} \models w \dot{\sqsubseteq} R$, if $w^{\mathcal{I}} \subseteq R^{\mathcal{I}}$. An interpretation is a *model* of a role hierarchy $R_h$ if it satisfies all RIAs in $R_h$, written $\mathcal{I} \models R_h$.

A RIA $w \dot{\sqsubseteq} R$ is $\prec$*-regular* if $R$ is a role name, and

1. $w = RR$, or
2. $w = R^-$, or
3. $w = S_1 \dots S_n$ and $S_i \prec R$, for all $1 \leq i \leq n$, or
4. $w = RS_1 \dots S_n$ and $S_i \prec R$, for all $1 \leq i \leq n$, or
5. $w = S_1 \dots S_n R$ and $S_i \prec R$, for all $1 \leq i \leq n$.

Finally, a role hierarchy $\mathcal{R}_h$ is *regular* if there exists a regular order $\prec$ such that each RIA in $\mathcal{R}_h$ is $\prec$-regular.

Regularity prevents a role hierarchy from containing cyclic dependencies. For instance, the role hierarchy

$$\{RS \dot{\sqsubseteq} S, \quad RT \dot{\sqsubseteq} R, \quad VT \dot{\sqsubseteq} T, \quad VS \dot{\sqsubseteq} V\}$$

is not regular because it would require $\prec$ to satisfy $S \prec V \prec T \prec R \prec S$, which would imply $S \prec S$, thus contradicting the irreflexivity of $\prec$. Such cyclic dependencies are known to lead to undecidability.

Also, note that RIAs of the form $RR^- \dot{\sqsubseteq} R$, which would imply (a weak form of) reflexivity of $R$, are not regular according to the definition of regular orderings. However, the same condition on $R$ can be imposed by using the GCI $\exists R.\top \dot{\sqsubseteq} \exists R.\mathsf{Self}$; see below.

From the definition of the semantics of inverse roles, it follows immediately that $(x, y) \in w^{\mathcal{I}}$ iff $(y, x) \in \mathsf{Inv}(w)^{\mathcal{I}}$. Hence, each model satisfying $w \dot{\sqsubseteq} S$ also satisfies $\mathsf{Inv}(w) \dot{\sqsubseteq} \mathsf{Inv}(S)$ (and vice versa), and thus the restriction to those RIAs with only role *names* on their right hand side does not have any effect on expressivity.

Given a role hierarchy $\mathcal{R}_h$, we define the relation $\dot{\sqsubseteq}^*$ to be the transitive-reflexive closure of $\dot{\sqsubseteq}$ over $\{R \dot{\sqsubseteq} S, \mathsf{Inv}(R) \dot{\sqsubseteq} \mathsf{Inv}(S) \mid R \dot{\sqsubseteq} S \in \mathcal{R}_h\}$. A role $R$ is called a *sub-role* (*super-role*) of a role $S$ if $R \dot{\sqsubseteq}^* S$ ($S \dot{\sqsubseteq}^* R$). Two roles $R$ and $S$ are *equivalent* ($R \equiv S$) if $R \dot{\sqsubseteq}^* S$ and $S \dot{\sqsubseteq}^* R$.

Note that, due to restriction (3) in the definition of $\prec$-regularity, we also restrict $\dot{\sqsubseteq}^*$ to be acyclic, and thus regular role hierarchies never contain two equivalent roles.[2]

Next, let us turn to the second component of Rboxes, the role assertions. For an interpretation $\mathcal{I}$, we define $Diag^{\mathcal{I}}$ to be the set $\{(x, x) \mid x \in \Delta^{\mathcal{I}}\}$. Note that, since the interpretation of the universal role $U$ is fixed in any given model (as the universal relation on $\Delta^I \times \Delta^I$ which is, by definition, reflexive, symmetric, and transitive), we disallow the universal role to appear in role assertions.

**Definition 5.3** [Role Assertions] For roles $R, S \neq U$, we call the assertions $\mathsf{Ref}(R)$, $\mathsf{Irr}(R)$, $\mathsf{Sym}(R)$, $\mathsf{Asy}(R)$, $\mathsf{Tra}(R)$, and $\mathsf{Dis}(R, S)$, *role assertions*, where, for each interpretation $\mathcal{I}$ and all $x,\ y,\ z \in \Delta^{\mathcal{I}}$, we have:

| | | |
|---|---|---|
| $\mathcal{I} \models \mathsf{Sym}(R)$ | if | $(x, y) \in R^{\mathcal{I}}$ implies $(y, x) \in R^I$; |
| $\mathcal{I} \models \mathsf{Asy}(R)$ | if | $(x, y) \in R^{\mathcal{I}}$ implies $(y, x) \notin R^{\mathcal{I}}$ |
| $\mathcal{I} \models \mathsf{Tra}(R)$ | if | $(x, y) \in R^{\mathcal{I}}$ and $(y, z) \in R^{\mathcal{I}}$ imply $(x, z) \in R^I$; |
| $\mathcal{I} \models \mathsf{Ref}(R)$ | if | $Diag^{\mathcal{I}} \subseteq R^{\mathcal{I}}$; |
| $\mathcal{I} \models \mathsf{Irr}(R)$ | if | $R^{\mathcal{I}} \cap Diag^{\mathcal{I}} = \emptyset$; |
| $\mathcal{I} \models \mathsf{Dis}(R, S)$ | if | $R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$. |

Adding symmetric and transitive role assertions is a trivial move since both of these expressive means can be replaced with complex role inclusion axioms as follows: $\mathsf{Sym}(R)$ is equivalent to $R^- \dot{\sqsubseteq} R$ and $\mathsf{Tra}(R)$ is equivalent to $RR \dot{\sqsubseteq} R$.

Thus, as far as expressivity is concerned, we can assume, for convenience, that no role assertions of the form $\mathsf{Tra}(R)$ or $\mathsf{Sym}(R)$ appear in $\mathcal{R}_a$, but that transitive and/or symmetric roles will be handled by the RIAs alone. In particular, notice that regularity of a role hierarchy is preserved when replacing such role assertions with the corresponding RIAs.

The situation is different, however, for the other Rbox assertions. None of reflexivity, irreflexivity, antisymmetry or disjointness of roles can be enforced by role inclusion axioms.

[2]This is not a serious restriction for, if $\mathcal{R}$ contains $\dot{\sqsubseteq}^*$ cycles, we can simply choose one role $R$ from each cycle and replace all other roles in this cycle with $R$ in the input Rbox, Tbox, and Abox.

However, as we shall see later, reflexivity and irreflexivity of roles are closely related to the new concept $\exists R.\mathsf{Self}$.

Note that the version of antisymmetry introduced above is *strict* in the sense that it also implies irreflexivity as opposed to the more widely used notion of antisymmetry which allows for reflexive points. For instance, in *mereology*, the relation `PartOf` is usually assumed to be 'reflexive antisymmetric' (i.e., reflexivity, plus $xRy$ and $yRx$ implies $x = y$), while the relation `properPartOf` is assumed to be 'irreflexive antisymmetric' (defined just as antisymmetry above)[Sim87, CV99]. The more general version of antisymmetry is more difficult to handle algorithmically, and we leave this to future work.

In $\mathcal{SHIQ}$, the application of qualified number restrictions has to be restricted to certain roles, called *simple roles*, to preserve decidability [HST99]. In the context of $\mathcal{SROIQ}$, the definition of *simple role* has to be slightly modified, and simple roles figure not only in qualified number restrictions, but in several other constructs as well. Intuitively, non-simple roles are those that are implied by the composition of roles.

Given a role hierarchy $\mathcal{R}_h$ and a set of role assertions $\mathcal{R}_a$ (without transitivity or symmetry assertions), the set of roles that are *simple in* $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ is inductively defined as follows:

- a role name is simple if it does not occur on the right hand side of a RIA in $\mathcal{R}_h$,
- an inverse role $R^-$ is simple if $R$ is, and
- if $R$ occurs on the right hand side of a RIA in $\mathcal{R}_h$, then $R$ is simple if, for each $w \dot{\sqsubseteq} R \in \mathcal{R}_h$, $w = S$ for a simple role $S$.

A set of role assertions $\mathcal{R}_a$ is called *simple* if all roles $R, S$ appearing in role assertions of the form $\mathsf{Irr}(R)$, $\mathsf{Asy}(R)$, or $\mathsf{Dis}(R, S)$, are simple in $\mathcal{R}$. If $\mathcal{R}$ is clear from the context, we often use "simple" instead of "simple in $\mathcal{R}$".

**Definition 5.4** [Role Box] A $\mathcal{SROIQ}$*-role box* (Rbox for short) is a set $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$, where $\mathcal{R}_h$ is a regular role hierarchy and $\mathcal{R}_a$ is a finite, simple set of role assertions.

An interpretation *satisfies a role box* $\mathcal{R}$ (written $\mathcal{I} \models \mathcal{R}$) if $\mathcal{I} \models R_h$ and $\mathcal{I} \models \phi$ for all role assertions $\phi \in R_a$. Such an interpretation is called a model of $\mathcal{R}$.

## 5.3 Concepts and Inference Problems for $\mathcal{SROIQ}$

**Definition 5.5** [$\mathcal{SROIQ}$ Concepts, Tboxes, and Aboxes]
The set of $\mathcal{SROIQ}$*-concepts* is the smallest set such that

- every concept name (including nominals) and $\top, \bot$ are concepts, and,
- if $C$, $D$ are concepts, $R$ is a role (possibly inverse), $S$ is a simple role (possibly inverse), and $n$ is a non-negative integer, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\exists S.\mathsf{Self}$, $(\geqslant n\ S\ C)$, and $(\leqslant n\ S\ C)$ are also concepts.

A *general concept inclusion axiom* (GCI) is an expression of the form $C \dot{\sqsubseteq} D$ for two $\mathcal{SROIQ}$-concepts $C$ and $D$. A *Tbox* $\mathcal{T}$ is a finite set of GCIs.

An *individual assertion* is of one of the following forms: $a:C$, $(a,b):R$, $(a,b):\neg R$, or $a \not\doteq b$, for $a, b \in \mathsf{N_N}$ (the set of individual names), a (possibly inverse) role $R$, and a $\mathcal{SROIQ}$-concept $C$. A $\mathcal{SROIQ}$-*Abox* $\mathcal{A}$ is a finite set of individual assertions.

It is part of future work to determine which of the restrictions to simple roles in role assertions $\mathsf{Dis}(R,S)$, $\mathsf{Asy}(R)$, and $\mathsf{Irr}(R)$, as well as the concept expression $\exists S.\mathsf{Self}$, are strictly necessary in order to preserve decidability or practicability. These restrictions, however, allow a rather smooth integration of the new constructs into existing algorithms.

**Definition 5.6** [Semantics and Inference Problems]

Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, concepts $C$, $D$, roles $R$, $S$, and non-negative integers $n$, the *extension of complex concepts* is defined inductively by the following equations, where $\sharp M$ denotes the cardinality of a set $M$, and concept names, roles, and nominals are interpreted as in Definition 5.1:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \qquad \bot^{\mathcal{I}} = \emptyset, \qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \\
(\exists R.\mathsf{Self})^{\mathcal{I}} &= \{x \mid (x,x) \in R^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\} \\
(\geqslant n\ R\ C)^{\mathcal{I}} &= \{x \mid \sharp\{y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\} \\
(\leqslant n\ R\ C)^{\mathcal{I}} &= \{x \mid \sharp\{y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\}
\end{aligned}$$

$\mathcal{I}$ is a *model of a Tbox* $\mathcal{T}$ (written $\mathcal{I} \models \mathcal{T}$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each GCI $C \dot{\sqsubseteq} D$ in $\mathcal{T}$. A concept $C$ is called *satisfiable* if there is an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$. A concept $D$ *subsumes* a concept $C$ (written $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each interpretation. For an interpretation $\mathcal{I}$, an element $x \in \Delta^{\mathcal{I}}$ is called an *instance* of a concept $C$ if $x \in C^{\mathcal{I}}$.

$\mathcal{I}$ *satisfies* (is a model of) *an Abox* $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) if for all individual assertions $\phi \in \mathcal{A}$ we have $\mathcal{I} \models \phi$, where

$$\begin{aligned}
&\mathcal{I} \models a{:}C && \text{if} \quad a^{\mathcal{I}} \in C^{\mathcal{I}}; \\
&\mathcal{I} \models a \not\doteq b && \text{if} \quad a^{\mathcal{I}} \neq b^{\mathcal{I}}; \\
&\mathcal{I} \models (a,b){:}R && \text{if} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}; \\
&\mathcal{I} \models (a,b){:}\neg R && \text{if} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}.
\end{aligned}$$

An Abox $\mathcal{A}$ is *consistent* with respect to an Rbox $\mathcal{R}$ and a Tbox $\mathcal{T}$ if there is a model $\mathcal{I}$ for $\mathcal{R}$ and $\mathcal{T}$ such that $\mathcal{I} \models \mathcal{A}$.

The above inference problems can be defined w.r.t. a role box $\mathcal{R}$ and/or a Tbox $\mathcal{T}$ in the usual way, i.e., by replacing *interpretation* with *model of* $\mathcal{R}$ and/or $\mathcal{T}$.

## 5.4 Reduction of Inference Problems

For DLs that are closed under negation, subsumption and (un)satisfiability of concepts can be mutually reduced: $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable, and $C$ is unsatisfiable iff $C \sqsubseteq \bot$. Furthermore, a concept $C$ is satisfiable iff the Abox $\{a{:}C\}$ ($a$ a 'new' individual name) is consistent.

It is straightforward to extend these reductions to Rboxes and Tboxes. In contrast, the reduction of inference problems w.r.t. a Tbox to pure concept inference problems (possibly w.r.t. a role hierarchy), deserves special care: in [Baa91, Sch91, BBN+93], the *internalisation* of GCIs is introduced, a technique that realises exactly this reduction. For $\mathcal{SROIQ}$, this technique only needs to be slightly modified. We will show in a series of steps that, in $\mathcal{SROIQ}$, satisfiability of a concept $C$ with respect to a triple $\langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ of, respectively, a $\mathcal{SROIQ}$ Abox, Rbox, and Tbox, can be reduced to concept satisfiability of a concept $C'$ with respect to an Rbox $\mathcal{R}'$, where the Rbox $\mathcal{R}'$ only contains role assertions of the form $\mathsf{Dis}(R, S)$, $\mathsf{Ref}(R)$, or $\mathsf{Asy}(R)$, and the universal role $U$ does not appear in $C'$.

While nominals can be used to 'internalise' the Abox, in order to eliminate the universal role, we use a 'simulated' universal role $U'$, i.e., a reflexive, symmetric, and transitive super-role of all roles and their inverses appearing in $\langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$, and which, additionally, connects all nominals appearing in the input.

Thus, let $C$ and $\langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ be a $\mathcal{SROIQ}$ concept and Abox, Rbox, and Tbox, respectively. In a first step, we replace the Abox $\mathcal{A}$ with an Abox $\mathcal{A}'$ such that $\mathcal{A}'$ only contains individual assertions of the form $a\!:\!C$. To this purpose, we associate with every individual $a \in \mathsf{N_N}$ appearing in $\mathcal{A}$ a new nominal $o_a$ not appearing in $\mathcal{T}$ or $C$. Next, $\mathcal{A}'$ is the result of replacing every individual assertion in $\mathcal{A}$ of the form $(a, b) : R$ with $a : \exists R.o_b$, every $(a, b) : \neg R$ with $a : \forall R.\neg o_b$, and every $a \neq b$ with $a : \neg o_b$. Now, given $C$ and $\mathcal{A}'$, define $C'$ as follows:

$$C' := C \sqcap \bigsqcap_{a:D \in \mathcal{A}'} \exists U.(o_a \sqcap D),$$

where $U$ is the universal role. It should be clear that C is satisfiable with respect to $\langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ if and only if $C'$ is satisfiable with respect to $\langle \mathcal{R}, \mathcal{T} \rangle$.

**Lemma 5.7 (Abox Elimination)** *$\mathcal{SROIQ}$ concept satisfiability with respect to Aboxes, Rboxes, and Tboxes is polynomially reducible to $\mathcal{SROIQ}$ concept satisfiability with respect to Rboxes and Tboxes only.*

Hence, in the following we will assume that Aboxes have been eliminated. Next, although we have the 'real' universal role $U$ present in the language, the following lemma shows how general concept inclusion axioms can be *internalised* while at the same time eliminating occurrences of the universal role $U$, using a simulated "universal" role $U'$, that is, a transitive super-role of all roles (except $U$) occurring in $\mathcal{T}$ or $\mathcal{R}$ and their respective inverses. Recall that the universal role $U$ is not allowed to appear in Rboxes.

**Lemma 5.8 (Tbox and Universal Role Elimination)** *Let $C$, $D$ be concepts, $\mathcal{T}$ a Tbox, and $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ an Rbox. Let $U' \neq U$ be a role that does not occur in $C$, $D$, $\mathcal{T}$, or $\mathcal{R}$, and, for $X$ a Tbox or a concept, let $X'$ result from $X$ by replacing every occurrence of $U$ with $U'$. We define*

$$C_{\mathcal{T}'} := \forall U'.\Big(\bigsqcap_{C_i' \dot{\sqsubseteq} D_i' \in \mathcal{T}'} \neg C_i' \sqcup D_i'\Big) \sqcap \Big(\bigsqcap_{\mathbf{N} \ni o \in \mathcal{T} \cup C \cup D} \exists U'.o\Big),$$

$$\mathcal{R}_h^{U'} := \mathcal{R}_h \cup \{R \dot{\sqsubseteq} U' \mid R \text{ occurs in } C', D', \mathcal{T}', \text{ or } \mathcal{R}\},$$

$$\mathcal{R}_a^{U'} := \mathcal{R}_a \cup \{\mathsf{Tra}(U'), \mathsf{Sym}(U'), \mathsf{Ref}(U')\}, \text{ and}$$

$$\mathcal{R}_{U'} := \mathcal{R}_h^{U'} \cup \mathcal{R}_a^{U'}. \text{ Then}$$

- *$C$ is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff $C' \sqcap C_{\mathcal{T}'}$ is satisfiable w.r.t. $\mathcal{R}_{U'}$.*
- *$D$ subsumes $C$ with respect to $\mathcal{T}$ and $\mathcal{R}$ iff $C' \sqcap \neg D' \sqcap C_{\mathcal{T}'}$ is unsatisfiable w.r.t. $\mathcal{R}_{U'}$.*

The proof of Lemma 5.8 is similar to the ones that can be found in [Sch91] and [Baa91]. Most importantly, it must be shown that (a): if a $\mathcal{SROIQ}$-concept $C$ is satisfiable with respect to a Tbox $\mathcal{T}$ and an Rbox $\mathcal{R}$, then $C, \mathcal{T}, \mathcal{R}$ have a *nominal connected* model, i.e., a model which is a union of connected components, where each such component contains a nominal, and where any two elements of a connected component are connected by a role path over those roles occurring in $C$, $\mathcal{T}$ or $\mathcal{R}$, and (b): if $y$ is reachable from $x$ via a role path (possibly involving inverse roles), then $(x, y) \in U'^{\mathcal{I}}$. These are easy consequences of the semantics and the definition of $U'$ and $C_{\mathcal{T}'}$, which guarantees that all nominals are connected by $U'$ links.

Now, note also that, instead of having a role assertion $\mathsf{Irr}(R) \in \mathcal{R}_a$, we can add, equivalently, the GCI $\top \dot{\sqsubseteq} \neg\exists R.\mathsf{Self}$ to $\mathcal{T}$, which can in turn be internalised. Likewise, instead of asserting $\mathsf{Ref}(R)$, we can, equivalently, add the GCI $\top \dot{\sqsubseteq} \exists R.\mathsf{Self}$ to $\mathcal{T}$. However, in the case of $\mathsf{Ref}(R)$ this replacement is only admissible for *simple* roles $R$ and thus not possible (syntactically) in general.

Thus, using these equivalences (including the replacement of Rbox assertions of the form $\mathsf{Sym}(R)$ and $\mathsf{Tra}(R)$) and Lemmas 5.7 and 5.8, we arrive at the following theorem:

**Theorem 5.9 (Reduction)**

1. *Satisfiability and subsumption of $\mathcal{SROIQ}$-concepts w.r.t. Tboxes, Aboxes, and Rboxes, are polynomially reducible to (un)satisfiability of $\mathcal{SROIQ}$-concepts w.r.t. Rboxes.*
2. *W.l.o.g., we can assume that Rboxes do not contain role assertions of the form $\mathsf{Irr}(R)$, $\mathsf{Tra}(R)$, or $\mathsf{Sym}(R)$, and that the universal role is not used.*

With Theorem 5.9, all standard inference problems for $\mathcal{SROIQ}$-concepts and Aboxes can be reduced to the problem of determining the consistency of a $\mathcal{SROIQ}$-concept w.r.t. to an Rbox (both not containing the universal role), where we can assume w.l.o.g. that all role assertions in the Rbox are of the form $\mathsf{Ref}(R)$, $\mathsf{Asy}(R)$, or $\mathsf{Dis}(R, S)$—we call such an Rbox *reduced*.

## 5.5 $\mathcal{SROIQ}$ is Decidable

In this section, we show that $\mathcal{SROIQ}$ is decidable. We present a tableau-based algorithm that decides the consistency of a $\mathcal{SROIQ}$ concept w.r.t. a reduced Rbox, and therefore also all standard inference problems as discussed above, see Theorem 5.9. Therefore, in the following, by Rbox we always mean *reduced* Rbox.

The algorithm tries to construct, for a $\mathcal{SROIQ}$-concept $C$ and an Rbox $\mathcal{R}$, a *tableau* for $C$ and $\mathcal{R}$, that is, an abstraction of a model of $C$ and $\mathcal{R}$.

For a regular role hierarchy $\mathcal{R}_h$ and a (possibly inverse) role $S$ occurring in $\mathcal{R}_h$, a non-deterministic finite automaton (NFA) $\mathcal{B}_S$ is defined. The construction of these automata

is identical to the one presented in [HS04], and we therefore refer to this paper for detailed definitions and proofs of the automata related results below.

The following proposition states that $\mathcal{B}_S$ indeed captures all implications between (paths of) roles and $S$ that are consequences of the role hierarchy $\mathcal{R}_h$, where $L(\mathcal{B}_S)$ denotes the language (a set of strings of roles) accepted by $\mathcal{B}_S$.

Unfortunately, as shown in [HS04], the size of $\mathcal{B}_R$ can be exponential in the size of $\mathcal{R}$. [HS04] consider certain further syntactic restrictions of role hierarchies (there called *simple* role hierarchies) that avoid this exponential blow-up. We conjecture that, without some such further restriction, this blow-up is unavoidable. The following technical Lemma from [HS04] will be needed later on.

**Lemma 5.10**

1. *$S \in L(\mathcal{B}_S)$ and, if $w \dot{\sqsubseteq} S \in \mathcal{R}$, then $w \in L(\mathcal{B}_S)$.*
2. *If $S$ is a simple role, then $L(\mathcal{B}_S) = \{R \mid R \sqsubseteq^* S\}$.*
3. *$L(\mathcal{B}_{\mathsf{Inv}(S)}) = \{\mathsf{Inv}(w) \mid w \in L(\mathcal{B}_S)\}$.*

### 5.5.1 A Tableau for $\mathcal{SROIQ}$

In the following, if not stated otherwise, $C, D$ (possibly with subscripts) denote $\mathcal{SROIQ}$-concepts (not using the universal role), $R, S$ (possibly with subscripts) roles, $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ an Rbox, and $\mathbf{R}_C$ the set of roles occurring in $C$ and $\mathcal{R}$ together with their inverses. Furthermore, as noted in Theorem 5.9, we can (and will from now on) assume w.l.o.g. that all role assertions appearing in $\mathcal{R}_a$ are of the form $\mathsf{Dis}(R, S)$, $\mathsf{Asy}(R)$, or $\mathsf{Ref}(R)$.

We start by defining $\mathsf{fclos}(C_0, \mathcal{R})$, the *closure* of a concept $C_0$ w.r.t. a regular role hierarchy $\mathcal{R}$. Intuitively, this contains all relevant sub-concepts of $C_0$ together with universal value restrictions over sets of role paths described by an NFA. We use NFAs in universal value restrictions to memorise the path between an object that has to satisfy a value restriction and other objects. To do this, we "push" this NFA-value restriction along all paths while the NFA gets "updated" with the path taken so far. For this "update", we use the following definition.

**Definition 5.11** For $\mathcal{B}$ an NFA and $q$ a state of $\mathcal{B}$, $\mathcal{B}(q)$ denotes the NFA obtained from $\mathcal{B}$ by making $q$ the (only) initial state of $\mathcal{B}$, and we use $q \xrightarrow{S} q' \in \mathcal{B}$ to denote that $\mathcal{B}$ has a transition $q \xrightarrow{S} q'$.

Without loss of generality, we assume all concepts to be in *negation normal form* (NNF), that is, negation occurs only in front of concept names or in front of $\exists R.\mathsf{Self}$. Any $\mathcal{SROIQ}$-concept can easily be transformed into an equivalent one in NNF by pushing negations inwards using a combination of De Morgan's laws and equivalences such as $\neg(\exists R.C) \equiv (\forall R.\neg C)$, $\neg(\leqslant n\ R\ C) \equiv (\geqslant (n+1)\ R\ C)$, etc. We use $\dot{\neg}C$ for the NNF of $\neg C$. Obviously, the length of $\dot{\neg}C$ is linear in the length of $C$.

For a concept $C_0$, $\mathsf{clos}(C_0)$ is the smallest set that contains $C_0$ and that is closed under sub-concepts and $\dot{\neg}$. The set $\mathsf{fclos}(C_0, \mathcal{R})$ is then defined as follows:

$$\mathsf{fclos}(C_0, \mathcal{R}) := \mathsf{clos}(C_0) \cup \{\forall \mathcal{B}_S(q).D \mid \forall S.D \in \mathsf{clos}(C_0) \text{ and } \mathcal{B}_S \text{ has a state } q\}.$$

It is not hard to show and well-known that the size of $\mathsf{clos}(C_0)$ is linear in the size of $C_0$. For the size of $\mathsf{fclos}(C_0, \mathcal{R})$, we have mentioned above that, for a role $S$, the size of $\mathcal{B}_S$ can be exponential in the depth of $\mathcal{R}$. Since there are at most linearly many concepts $\forall S.D$, this yields a bound for the cardinality of $\mathsf{fclos}(C_0, \mathcal{R})$ that is exponential in the depth of $\mathcal{R}$ and linear in the size of $C_0$.

**Definition 5.12** [Tableau] $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a *tableau* for $C_0$ w.r.t. $\mathcal{R}$ if

- $\mathbf{S}$ is a non-empty set;
- $\mathcal{L} : \mathbf{S} \to 2^{\mathsf{fclos}(C_0,\mathcal{R})}$ maps each element in $\mathbf{S}$ to a set of concepts;
- $\mathcal{E} : \mathbf{R}_{C_0} \to 2^{\mathbf{S}\times\mathbf{S}}$ maps each role to a set of pairs of elements in $\mathbf{S}$;
- $C_0 \in \mathcal{L}(s)$ for some $s \in \mathbf{S}$.

Furthermore, for all $s, t \in \mathbf{S}$, $C, C_1, C_2 \in \mathsf{fclos}(C_0, \mathcal{R})$, $o \in \mathbf{N} \cap \mathsf{fclos}(C_0, \mathcal{R})$, $R, S \in \mathbf{R}_{C_0}$, and

$$S^T(s, C) := \{r \in \mathbf{S} \mid (s, r) \in \mathcal{E}(S) \text{ and } C \in \mathcal{L}(r)\},$$

the tableau $T$ satisfies:

(P1) $C \in \mathcal{L}(s) \implies \neg C \notin \mathcal{L}(s)$, ($C$ atomic or $\exists R.\mathsf{Self}$);

(P2) $\top \in \mathcal{L}(s)$, and $\bot \notin \mathcal{L}(s)$;

(P3) $\exists R.\mathsf{Self} \in \mathcal{L}(s) \implies (s, s) \in \mathcal{E}(R)$;

(P4) $\neg\exists R.\mathsf{Self} \in \mathcal{L}(s) \implies (s, s) \notin \mathcal{E}(R)$;

(P5) $C_1 \sqcap C_2 \in \mathcal{L}(s) \implies C_1, C_2 \in \mathcal{L}(s)$;

(P6) $C_1 \sqcup C_2 \in \mathcal{L}(s) \implies C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$;

(P7) $\forall \mathcal{B}(p).C \in \mathcal{L}(s)$, $(s, t) \in \mathcal{E}(S)$, and $p \xrightarrow{S} q \in \mathcal{B}(p) \implies \forall \mathcal{B}(q).C \in \mathcal{L}(t)$;

(P8) $\forall \mathcal{B}.C \in \mathcal{L}(s)$ and $\varepsilon \in L(\mathcal{B}) \implies C \in \mathcal{L}(s)$;

(P9) $\forall S.C \in \mathcal{L}(s) \implies \forall \mathcal{B}_S.C \in \mathcal{L}(s)$;

(P10) $\exists S.C \in \mathcal{L}(s) \implies$ there is some $r \in \mathbf{S}$ with $(s, r) \in \mathcal{E}(S)$ and $C \in \mathcal{L}(r)$;

(P11) $(s, t) \in \mathcal{E}(R) \iff (t, s) \in \mathcal{E}(\mathsf{Inv}(R))$;

(P12) $(s, t) \in \mathcal{E}(R)$ and $R \sqsubseteq^* S \implies (s, t) \in \mathcal{E}(S)$;

(P13) $(\leqslant n\ S\ C) \in \mathcal{L}(s) \implies \sharp S^T(s, C) \leqslant n$;

(P14) $(\geqslant n\ S\ C) \in \mathcal{L}(s) \implies \sharp S^T(s, C) \geqslant n$;

(P15) $(\leqslant n\ S\ C) \in \mathcal{L}(s)$ and $(s, t) \in \mathcal{E}(S) \implies C \in \mathcal{L}(t)$ or $\dot{\neg} C \in \mathcal{L}(t)$;

(P16) $\mathsf{Dis}(R, S) \in \mathcal{R}_a \implies \mathcal{E}(R) \cap \mathcal{E}(S) = \emptyset$;

(P17) $\mathsf{Ref}(R) \in \mathcal{R}_a \implies (s,s) \in \mathcal{E}(R)$;

(P18) $\mathsf{Asy}(R) \in \mathcal{R}_a \implies (s,t) \in \mathcal{E}(R)$ implies $(t,s) \notin \mathcal{E}(R)$;

(P19) $o \in \mathcal{L}(r)$ for some $r \in \mathbf{S}$;

(P20) $o \in \mathcal{L}(s) \cap \mathcal{L}(t) \implies s = t$.

**Theorem 5.13 (Tableau)** *A $\mathcal{SROIQ}$-concept $C_0$ is satisfiable w.r.t. a reduced Rbox $\mathcal{R}$ iff there exists a tableau for $C_0$ w.r.t. $\mathcal{R}$.*

### 5.5.2 The Tableau Algorithm

In this section, we present a terminating, sound, and complete tableau algorithm that decides consistency of $\mathcal{SROIQ}$-concepts not using the universal role w.r.t. reduced Rboxes, and thus, using Theorem 5.9, also concept satisfiability w.r.t. Rboxes, Tboxes and Aboxes.

We first define the underlying data structures and corresponding operations. For more detailed explanations concerning the intuitions underlying these definitions, consult [HS05].

The algorithm generates a *completion graph*, a structure that, if complete and clash-free, can be unravelled to a (possibly infinite) tableau for the input concept and Rbox. Moreover, it is shown that the algorithm returns a complete and clash-free completion graph for $C_0$ and $\mathcal{R}$ if and only if there exists a tableau for $C_0$ and $\mathcal{R}$, and thus with Lemma 5.13, if and only if the concept $C_0$ is satisfiable w.r.t. $\mathcal{R}$.

As usual, in the presence of transitive roles, *blocking* is employed to ensure termination of the algorithm [HST00].

**Definition 5.14** [Completion Graph] Let $\mathcal{R}$ be a reduced Rbox, let $C_0$ be a $\mathcal{SROIQ}$-concept in NNF not using the universal role, and let $\mathbf{N}$ be the set of nominals. A *completion graph* for $C_0$ with respect to $\mathcal{R}$ is a directed graph $\mathcal{G} = (V, E, \mathcal{L}, \neq)$ where each node $x \in V$ is labelled with a set

$$\mathcal{L}(x) \subseteq \mathsf{fclos}(C_0, \mathcal{R}) \cup \mathbf{N} \cup \{(\leqslant m\ R\ C) \mid (\leqslant n\ R\ C) \in \mathsf{fclos}(C_0, \mathcal{R}) \text{ and } m \leq n\}$$

and each edge $(x,y) \in E$ is labelled with a set of role names $\mathcal{L}((x,y))$ containing (possibly inverse) roles occurring in $C_0$ or $\mathcal{R}$. Additionally, we keep track of inequalities between nodes of the graph with a symmetric binary relation $\neq$ between the nodes of $\mathcal{G}$.

In the following, we often use $R \in \mathcal{L}((x,y))$ as an abbreviation for $(x,y) \in E$ and $R \in \mathcal{L}((x,y))$.

If $(x,y) \in E$, then $y$ is called a *successor* of $x$ and $x$ is called a *predecessor* of $y$. *Ancestor* is the transitive closure of predecessor, and *descendant* is the transitive closure of successor. A node $y$ is called an *$R$-successor* of a node $x$ if, for some $R'$ with $R' \sqsubseteq^* R$, $R' \in \mathcal{L}((x,y))$. A node $y$ is called a *neighbour* (*$R$-neighbour*) of a node $x$ if y is a successor ($R$-successor) of $x$ or if $x$ is a successor ($\mathsf{Inv}(R)$-successor) of $y$.

For a role $S$ and a node $x$ in $\mathcal{G}$, we define the set of $x$'s $S$-neighbours with $C$ in their label, $S^{\mathcal{G}}(x,C)$, as follows:

$$S^{\mathcal{G}}(x,C) := \{y \mid y \text{ is an } S\text{-neighbour of } x \text{ and } C \in \mathcal{L}(y)\}.$$

$\mathcal{G}$ is said to *contain a clash* if there are nodes $x$ and $y$ such that

1. $\bot \in \mathcal{L}(x)$, or
2. for some concept name $A$, $\{A, \neg A\} \subseteq \mathcal{L}(x)$, or
3. $x$ is an $S$-neighbour of $x$ and $\neg\exists S.\mathsf{Self} \in \mathcal{L}(x)$, or
4. there is some $\mathsf{Dis}(R, S) \in \mathcal{R}_a$ and $y$ is an $R$- and an $S$-neighbour of $x$, or
5. there is some $\mathsf{Asy}(R) \in \mathcal{R}_a$ and $y$ is an $R$-neighbour of $x$ and $x$ is an $R$-neighbour of $y$, or
6. there is some concept $(\leqslant n\ S\ C) \in \mathcal{L}(x)$ and $\{y_0, \ldots, y_n\} \subseteq S^{\mathcal{G}}(x, C)$ with $y_i \neq\!\!\!\!\cdot\ y_j$ for all $0 \leq i < j \leq n$, or
7. for some $o \in \mathbf{N}$, $x \neq\!\!\!\!\cdot\ y$ and $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$.

If $o_1, \ldots, o_\ell$ are all the nominals occurring in $C_0$ then the tableau algorithm is initialised with the completion graph $\mathcal{G} = (\{r_0, r_1 \ldots, r_\ell\}, \emptyset, \mathcal{L}, \emptyset)$ with $\mathcal{L}(r_0) = \{C_0\}$ and $\mathcal{L}(r_i) = \{o_i\}$ for $1 \leq i \leq \ell$. $\mathcal{G}$ is then expanded by repeatedly applying the expansion rules given in Figure 2, stopping if a clash occurs.

Before describing the tableau algorithm in more detail, we define some terms and operations used in the (application of the) expansion rules:

**Nominal Nodes and Blockable Nodes** A node $x$ is a *nominal node* if $\mathcal{L}(x)$ contains a nominal. A node that is not a nominal node is a *blockable* node. A nominal $o \in \mathbf{N}$ is said to be *new in* $\mathcal{G}$ if no node in $\mathcal{G}$ has $o$ in its label.

**Blocking** A node $x$ is *label blocked* if it has ancestors $x'$, $y$ and $y'$ such that

1. $x$ is a successor of $x'$ and $y$ is a successor of $y'$,
2. $y$, $x$ and all nodes on the path from $y$ to $x$ are blockable,
3. $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$, *and*
4. $\mathcal{L}((x', x)) = \mathcal{L}((y', y))$.

In this case, we say that $y$ *blocks* $x$. A node is *blocked* if either it is label blocked or it is blockable and its predecessor is blocked; if the predecessor of a blockable node $x$ is blocked, then we say that $x$ is *indirectly blocked*.

**Generating and Shrinking Rules and Safe Neighbours** The $\geqslant$-, $\exists$- and $NN$-rules are called *generating rules*, and the $\leqslant$- and the $o$-rule are called *shrinking rules*. An $R$-neighbour $y$ of a node $x$ is *safe* if (i) $x$ is blockable or if (ii) $x$ is a nominal node and $y$ is not blocked.

**Pruning** When a node $y$ is *merged* into a node $x$, we "prune" the completion graph by removing $y$ and, recursively, all blockable successors of $y$. More precisely, pruning a node $y$ (written $\mathsf{Prune}(y)$) in $\mathcal{G} = (V, E, \mathcal{L}, \neq\!\!\!\!\cdot)$ yields a graph that is obtained from $\mathcal{G}$ as follows:

1. for all successors $z$ of $y$, remove $(y, z)$ from $E$ and, if $z$ is blockable, $\mathsf{Prune}(z)$;
2. remove $y$ from $V$.

**Merging** In some rules, we "merge" one node into another node. Intuitively, when we merge a node $y$ into a node $x$, we add $\mathcal{L}(y)$ to $\mathcal{L}(x)$, "move" all the edges leading *to* $y$ so that they lead to $x$ and "move" all the edges leading from $y$ to nominal nodes so that they lead from $x$ to the same nominal nodes; we then remove $y$ (and blockable sub-trees below $y$) from the completion graph. More precisely, merging a node $y$ into a node $x$ (written $\mathsf{Merge}(y, x)$) in $\mathcal{G} = (V, E, \mathcal{L}, \neq)$ yields a graph that is obtained from $\mathcal{G}$ as follows:

1. for all nodes $z$ such that $(z, y) \in E$
   (a) if $\{(x, z), (z, x)\} \cap E = \emptyset$, then add $(z, x)$ to $E$ and set $\mathcal{L}((z, x)) = \mathcal{L}((z, y))$,
   (b) if $(z, x) \in E$, then set $\mathcal{L}((z, x)) = \mathcal{L}((z, x)) \cup \mathcal{L}((z, y))$,
   (c) if $(x, z) \in E$, then set $\mathcal{L}((x, z)) = \mathcal{L}((x, z)) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}((z, y))\}$, and
   (d) remove $(z, y)$ from $E$;
2. for all nominal nodes $z$ such that $(y, z) \in E$
   (a) if $\{(x, z), (z, x)\} \cap E = \emptyset$, then add $(x, z)$ to $E$ and set $\mathcal{L}((x, z)) = \mathcal{L}((y, z))$,
   (b) if $(x, z) \in E$, then set $\mathcal{L}((x, z)) = \mathcal{L}((x, z)) \cup \mathcal{L}((y, z))$,
   (c) if $(z, x) \in E$, then set $\mathcal{L}((z, x)) = \mathcal{L}((z, x)) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}((y, z))\}$, and
   (d) remove $(y, z)$ from $E$;
3. set $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$;
4. add $x \neq z$ for all $z$ such that $y \neq z$; and
5. $\mathsf{Prune}(y)$.

If $y$ was merged into $x$, we call $x$ a *direct heir* of $y$, and we use being an *heir* of another node for the transitive closure of being a "direct heir".

**Level (of Nominal Nodes)** Let $o_1, \ldots, o_\ell$ be all the nominals occurring in the input concept $D$. We define the *level* of a node inductively as follows:

- each (nominal) node $x$ with an $o_i \in \mathcal{L}(x)$, $1 \leq i \leq \ell$, is of level 0, and
- a nominal node $x$ is of level $i$ if $x$ is not of some level $j < i$ and $x$ has a neighbour that is of level $i - 1$.

**Strategy (of Rule Application)** The expansion rules in Figure 2 are applied according to the following strategy:

1. the $o$-rule is applied with highest priority,
2. next, the $\leqslant$- and the $NN$-rule are applied, and they are applied first to nominal nodes with lower levels (before they are applied to nodes with higher levels). In case they are both applicable to the same node, the $NN$-rule is applied first.

3. all other rules are applied with a lower priority.

We are now ready to finish the description of the tableau algorithm. A completion graph is *complete* if it contains a clash, or when none of the rules is applicable. If the expansion rules can be applied to $C_0$ and $\mathcal{R}$ in such a way that they yield a complete, clash-free completion graph, then the algorithm returns "$C_0$ is *satisfiable* w.r.t. $\mathcal{R}$", and "$C_0$ is *unsatisfiable* w.r.t. $\mathcal{R}$" otherwise.

As usual, we prove termination, soundness, and completeness of the tableau algorithm to show that it indeed decides satisfiability of $\mathcal{SROIQ}$-concepts w.r.t. Rboxes.

**Theorem 5.15 (Termination, Soundness, and Completeness)**
*Let $C_0$ be a $\mathcal{SROIQ}$-concept in NNF and $\mathcal{R}$ a reduced Rbox.*

1. *The tableau algorithm terminates when started with $C_0$ and $\mathcal{R}$.*
2. *The expansion rules can be applied to $C_0$ and $\mathcal{R}$ such that they yield a complete and clash-free completion graph if and only if there is a tableau for $C_0$ w.r.t. $\mathcal{R}$.*

From Theorems 5.9, 5.13 and 5.15, we thus arrive at the following theorem:

**Theorem 5.16 (Decidability)** *The tableau algorithm decides satisfiability and subsumption of $\mathcal{SROIQ}$-concepts with respect to Aboxes, Rboxes, and Tboxes.*

## 5.6 Outlook

In this section, we have introduced a description logic, $\mathcal{SROIQ}$, that overcomes certain shortcomings in expressiveness of other DLs. We have used $\mathcal{SHOIQ}$ as a starting point, extended it with "useful-yet-harmless" expressive means, and extended the tableau algorithm accordingly. $\mathcal{SROIQ}$ is intended to be a basis for future extensions of OWL, and has already been adopted as the logical basis of OWL 1.1.

It is left for future work to determine whether the restrictions to simple roles can be relaxed, to pinpoint the exact computational complexity of $\mathcal{SROIQ}$, and to include further role assertions such as the more general version of antisymmetry to allow a better modeling of mereological notions.

A further line of investigation concerns concrete datatypes with inverse functional datatype properties: these are of interest since they allow to express simple key constraints. For instance, we might want to use a datatype property `SSN` for social security number as a key for US citizen.

Another line of research within the TONES project that investigates expressive DLs and has led to very interesting new decidability results is concerned with DLs that are related to propositional dynamic logic with intersection. We omit a detailed discussion of these logics here, and refer e.g. to the TONES paper [GLL07].

| | |
|---|---|
| $\sqcap$-rule: | if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$,<br>then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$ |
| $\sqcup$-rule: | if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$<br>then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$ |
| $\exists$-rule: | if $\exists S.C \in \mathcal{L}(x)$, $x$ is not blocked, and $x$ has no $S$-neighbour $y$ with $C \in \mathcal{L}(y)$<br>then create a new node $y$ with $\mathcal{L}((x,y)) := \{S\}$ and $\mathcal{L}(y) := \{C\}$ |
| Self–Ref-rule: | if $\exists S.\mathsf{Self} \in \mathcal{L}(x)$ or $\mathsf{Ref}(S) \in \mathcal{R}_a$, $x$ is not blocked, and $S \notin \mathcal{L}((x,x))$<br>then add an edge $(x,x)$ if it does not yet exist, and set $\mathcal{L}((x,x)) \longrightarrow \mathcal{L}((x,x)) \cup \{S\}$ |
| $\forall_1$-rule: | if $\forall S.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and $\forall \mathcal{B}_S.C \notin \mathcal{L}(x)$<br>then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\forall \mathcal{B}_S.C\}$ |
| $\forall_2$-rule: | if $\forall \mathcal{B}(p).C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, $p \xrightarrow{S} q$ in $\mathcal{B}(p)$, and there is an $S$-neighbour $y$ of $x$ with $\forall \mathcal{B}(q).C \notin \mathcal{L}(y)$,<br>then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall \mathcal{B}(q).C\}$ |
| $\forall_3$-rule: | if $\forall \mathcal{B}.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, $\varepsilon \in L(\mathcal{B})$ and $C \notin \mathcal{L}(x)$<br>then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$ |
| choose-rule: | if $(\leqslant n\ S\ C) \in \mathcal{L}(x)$, $x$ not indirectly blocked, and<br>exists $S$-neighbour $y$ of $x$ with $\{C, \doteq\neg C\} \cap \mathcal{L}(y) = \emptyset$<br>then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \doteq\neg C\}$ |
| $\geqslant$-rule: | if 1. $(\geqslant n\ S\ C) \in \mathcal{L}(x)$, $x$ is not blocked<br>2. there are not $n$ safe $S$-neighbours $y_1, \ldots, y_n$ of $x$ with $C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$<br>then create $n$ new nodes $y_1, \ldots, y_n$ with $\mathcal{L}((x,y_i)) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$. |
| $\leqslant$-rule: | if 1. $(\leqslant n\ S\ C) \in \mathcal{L}(z)$, $z$ is not indirectly blocked<br>2. $\sharp S^{\mathcal{G}}(z, C) > n$ and there are two $S$-neighbours $x, y$ of $z$ with $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$, and not $x \neq y$<br>then 1. if $x$ is a nominal node then $\mathsf{Merge}(y, x)$<br>2. else, if $y$ is a nominal node or an ancestor of $x$ then $\mathsf{Merge}(x, y)$<br>3. else $\mathsf{Merge}(y, x)$ |
| $o$-rule: | if for some $o \in N_I$ there are 2 nodes $x, y$ with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \neq y$<br>then $\mathsf{Merge}(x, y)$ |
| $NN$-rule: | if 1. $(\leqslant n\ S\ C) \in \mathcal{L}(x)$, $x$ is a nominal node, and there is a blockable $S$-neighbour $y$ of $x$ such that $C \in \mathcal{L}(y)$ and $x$ is a successor of $y$,<br>2. there is no $m$ such that $1 \leqslant m \leqslant n$, $(\leqslant m\ S\ C) \in \mathcal{L}(x)$, and there exist $m$ nominal $S$-neighbours $z_1, \ldots, z_m$ of $x$ with $C \in \mathcal{L}(z_i)$ and $z_i \neq z_j$ for all $1 \leq i < j \leq m$.<br>then 1. guess $m$ with $1 \leqslant m \leqslant n$, and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leqslant m\ S\ C)\}$<br>2. create $m$ new nodes $y_1, \ldots, y_m$ with $\mathcal{L}((x,y_i)) = \{S\}$, $\mathcal{L}(y_i) = \{C, o_i\}$, for each $o_i \in N_I$ new in $\mathcal{G}$, and $y_i \neq y_j$ for $1 \leq i < j \leq m$. |

Figure 2: The Expansion Rules for the $\mathcal{SROIQ}$ Tableau Algorithm.

# 6 Description Logics with Concrete Domains

## 6.1 Introduction

To use description logics (DLs) in an application, it is crucial to identify a DL that is sufficiently expressive to represent the relevant notions of the application domain, but for which reasoning is still decidable.

For several relevant applications of DL-based ontologies such as the semantic web and reasoning about ER and UML diagrams, there is a need for DLs that include, among others, the expressive means *concrete domains* and *general TBoxes* [BHS03b, CLN98, Lut02b]. The purpose of concrete domains is to enable the definition of concepts with reference to concrete qualities of their instances such as the weight, age, duration, and spatial extension. General TBoxes play an important role in modern DLs as they allow to represent background knowledge of application domains by stating via inclusions $C \sqsubseteq D$ that the extension of a concept $C$ is included in the extension of a concept $D$.

Unfortunately, combining concrete domains with general TBoxes easily leads to undecidability. For example, it has been shown in [Lut04b] that the basic DL $\mathcal{ALC}$ extended with general TBoxes and a rather inexpressive concrete domain based on the natural numbers and providing for equality and incrementation predicates is undecidable, see also the survey paper [Lut03]. In view of this discouraging result, it is a natural question whether there are *any* useful concrete domains that can be combined with general TBoxes in a decidable DL. A positive answer to this question has been given in [Lut04a] and [Lut02a], where two such well-behaved concrete domains are identified: a temporal one based on the Allen relations for interval-based temporal reasoning, and a numerical one based on the reals and equipped with various unary and binary predicates such as "$\leq$", "$>_5$", and "$\neq$". Using an automata-based approach, it has been shown in [Lut04a, Lut02a] that reasoning in the DLs $\mathcal{ALC}$ and $\mathcal{SHIQ}$ extended with these concrete domains and general TBoxes is decidable and ExpTime-complete.

In this section, we summarize the TONES publication [LM07], whose contribution is two-fold: first, instead of focusing on particular concrete domains as in previous work, we identify a *general* property of concrete domains, called $\omega$-admissibility, that is sufficient for proving decidability of DLs equipped with concrete domains and general TBoxes. For defining $\omega$-admissibility, we concentrate on a particular kind of concrete domains: *constraint systems*. Roughly, a constraint system is a concrete domain that only has binary predicates, which are interpreted as jointly exhaustive and pairwise disjoint (JEPD) relations. We exhibit two example constraint systems that are $\omega$-admissible: a temporal one based on the real line and the Allen relations [All83], and a spatial one based on the real plane and the RCC8 relations [EF91, Ben97, RCC92].

Second, we develop a *tableau algorithm* for DLs with both general TBoxes and concrete domains. This algorithm is used to establish a general decidability result for the concept satisfiability and subsumption problem[3] in $\mathcal{ALC}$ equipped with general TBoxes and any $\omega$-admissible concrete domain. In particular, we obtain decidability of $\mathcal{ALC}$ with general TBoxes and the Allen relations as first established in [Lut04a], and, as a new result, prove decidability of $\mathcal{ALC}$ with general TBoxes and the RCC8 relations as a concrete

[3] for the detailed description of the problem see Section 3.3 of [LLS06]

domain. In contrast to existing tableau algorithms [HMW01, HS01], we do not impose any restrictions on the concrete domain constructor. As state-of-the-art DL reasoners such as FaCT and RACER are based on tableau algorithms similar to the one described in this section [Hor98, HM01], we view our algorithm as a first step towards an efficient implementation of description logics with ($\omega$-admissible) concrete domains and general TBoxes. In particular, we identify an expressive fragment of our logic that should be easily integrated into existing DL reasoners.

Therefore, we are interested in stand-alone ontologies and mainly address the ontology design task of authoring concept descriptions and of structuring the ontology.

## 6.2 Constraint Systems

We introduce a general notion of *constraint system* that is intended to capture standard constraint systems based on a set of jointly-exhaustive and pairwise-disjoint (JEPD) binary relations. Examples for such systems include spatial constraint networks based on the RCC8 relations [EF91, Ben97, RN99] or on cardinal direction relations [Fra96], and temporal constraint networks based on Allen's relations of time intervals [All83, VKvB90, NB95] or on relations between time points [VK86, VKvB90].

**Definition 6.1** [Rel-network] Let Var be a countably infinite set of variables and Rel a finite set of binary relation symbols. A Rel-*constraint* is an expression $(x\ r\ y)$ with $x, y \in \mathsf{Var}$ and $r \in \mathsf{Rel}$. A Rel-*network* is a (finite or infinite) set of Rel-constraints. For $N$ a Rel-network, we use $V_N$ to denote the variables used in $N$. We say that $N$ is *complete* if, for all $x, y \in V_N$, there is exactly one constraint $(x\ r\ y) \in N$.

We define the semantics of Rel-network by using complete Rel-networks as models. Intuitively, the nodes in these complete networks should be viewed as concrete values rather than as variables. Equivalently to our network-based semantics, we could proceed as in constraint satisfaction problems, associate each variable with a set of values, and view relations as constraints on these values, see e.g. [RN95].

**Definition 6.2** [Model, Constraint System] Let $N$ be a Rel-network and $N'$ a complete Rel-networks. We say that $N'$ is a *model* of $N$ if there is a mapping $\tau : V_N \to V_{N'}$ such that $(x\ r\ y) \in N$ implies $(\tau(x)\ r\ \tau(y)) \in N'$.

A *constraint system* $\mathcal{C} = \langle \mathsf{Rel}, \mathfrak{M} \rangle$ consists of a finite set of binary relation symbols Rel and a set $\mathfrak{M}$ of complete Rel-networks (the *models* of $\mathcal{C}$). A Rel-network $N$ is *satisfiable* in $\mathcal{C}$ if $\mathfrak{M}$ contains a model of $N$.

To emphasize the different role of variables in Rel-networks and in models, we denote variables in the former with $x, y, \ldots$ and in the latter with $v, v'$, etc. Note that Rel-networks used as models have to be complete, which corresponds to the relations in Rel to be jointly exhaustive and mutually exclusive.

Equivalently to our network-based semantics, we could proceed as in constraint satisfaction problems, associate each variable with a set of values, and view relations as constraints on these values, see e.g. [RN95].

In the following two subsections, we introduce two example constraint systems: one for spatial reasoning based on the RCC8 topological relations in the real plane, and one for temporal reasoning based on the Allen relations in the real line.

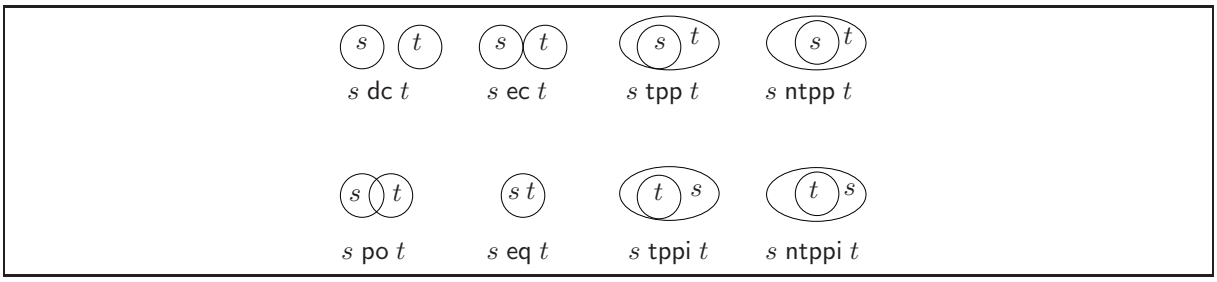


Figure 3: The eight RCC8 relations.

### 6.2.1 RCC8

The RCC8 relations, which are illustrated in Figure 3, are intended to describe the relation between regions in topological spaces [RCC92]. Here, we will use the standard topology of the real plane which is one of the most appropriate topologies for spatial reasoning. Let

$$\mathsf{RCC8} = \{\mathsf{eq}, \mathsf{dc}, \mathsf{ec}, \mathsf{po}, \mathsf{tpp}, \mathsf{ntpp}, \mathsf{tppi}, \mathsf{ntppi}\}$$

denote the RCC8 relations. Recall that a topological space is a pair $\mathfrak{T} = (U, \mathbb{I})$, where $U$ is a set and $\mathbb{I}$ is an *interior operator* on $U$, i.e., for all $s, t \subseteq U$, we have

$$\begin{array}{rclcrcl} \mathbb{I}(U) & = & U & & \mathbb{I}(s) & \subseteq & s \\ \mathbb{I}(s) \cap \mathbb{I}(t) & = & \mathbb{I}(s \cap t) & & \mathbb{II}(s) & = & \mathbb{I}(s). \end{array}$$

As usual, the closure operator $\mathbb{C}$ is defined as $\mathbb{C}(s) = \overline{\mathbb{I}(\overline{s})}$, where $\overline{t} = U \setminus t$, for $t \subseteq U$. As the *regions* of a topological space $\mathfrak{T} = (U, \mathbb{I})$, we use the set of non-empty, regular closed subsets of $U$, where a subset $s \subseteq U$ is called *regular closed* if $\mathbb{CI}(s) = s$. Given a topological space $\mathfrak{T}$ and a set of regions $U_{\mathfrak{T}}$, we define the extension of the RCC8 relations as the following subsets of $U_{\mathfrak{T}} \times U_{\mathfrak{T}}$:

$$\begin{array}{rcl} (s,t) \in \mathsf{dc}^{\mathfrak{T}} & \text{iff} & s \cap t = \emptyset \\ (s,t) \in \mathsf{ec}^{\mathfrak{T}} & \text{iff} & \mathbb{I}(s) \cap \mathbb{I}(t) = \emptyset \ \wedge \ s \cap t \neq \emptyset \\ (s,t) \in \mathsf{po}^{\mathfrak{T}} & \text{iff} & \mathbb{I}(s) \cap \mathbb{I}(t) \neq \emptyset \ \wedge \ s \setminus t \neq \emptyset \ \wedge t \setminus s \neq \emptyset \\ (s,t) \in \mathsf{eq}^{\mathfrak{T}} & \text{iff} & s = t \\ (s,t) \in \mathsf{tpp}^{\mathfrak{T}} & \text{iff} & s \cap \overline{t} = \emptyset \ \wedge \ s \cap \overline{\mathbb{I}(t)} \neq \emptyset \wedge s \neq t \\ (s,t) \in \mathsf{ntpp}^{\mathfrak{T}} & \text{iff} & s \cap \overline{\mathbb{I}(t)} = \emptyset \wedge s \neq t \\ (s,t) \in \mathsf{tppi}^{\mathfrak{T}} & \text{iff} & (t,s) \in \mathsf{tpp}^{\mathfrak{T}} \\ (s,t) \in \mathsf{ntppi}^{\mathfrak{T}} & \text{iff} & (t,s) \in \mathsf{ntpp}^{\mathfrak{T}}. \end{array}$$

Let $\mathfrak{T}_{\mathbb{R}^2}$ be the standard topology on $\mathbb{R}^2$ induced by the Euclidean metric, and let $\mathcal{RS}_{\mathbb{R}^2}$ be the set of all non-empty regular closed subsets of $\mathfrak{T}_{\mathbb{R}^2}$. Then we define the constraint system

$$\mathsf{RCC8}_{\mathbb{R}^2} = \langle \mathsf{RCC8}, \mathfrak{M}_{\mathbb{R}^2} \rangle$$

by setting $\mathfrak{M}_{\mathbb{R}^2} := \{N_{\mathbb{R}^2}\}$, where $N_{\mathbb{R}^2}$ is defined by fixing a variable $v_s \in \mathsf{Var}$ for every $s \in \mathcal{RS}_{\mathbb{R}^2}$ and setting

$$N_{\mathbb{R}^2} := \{(v_s \ r \ v_t) \mid r \in \mathsf{RCC8}, \ s, t \in \mathcal{RS}_{\mathbb{R}^2} \text{ and } (s,t) \in r^{\mathfrak{T}_{\mathbb{R}^2}}\}.$$

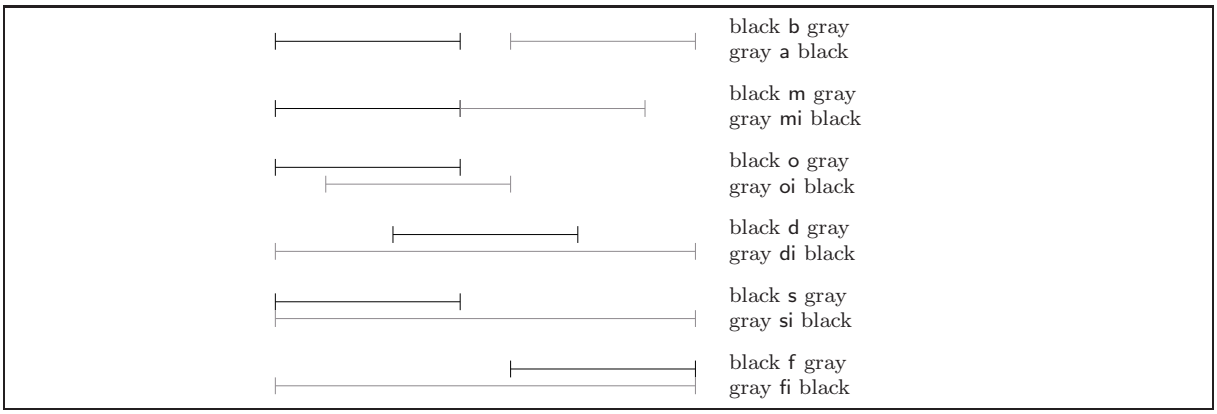


Figure 4: The thirteen Allen relations.

Note that using only regular closed sets excludes sub-dimensional regions such as points and lines. This is necessary for the RCC8 relations to be jointly exhaustive and pairwise disjoint.

### 6.2.2 Allen's Relations

In artificial intelligence, constraint systems based on Allen's interval relations are a popular tool for the representation of temporal knowledge [All83]. Let

$$\mathsf{Allen} = \{\mathsf{b}, \mathsf{a}, \mathsf{m}, \mathsf{mi}, \mathsf{o}, \mathsf{oi}, \mathsf{d}, \mathsf{di}, \mathsf{s}, \mathsf{si}, \mathsf{f}, \mathsf{fi}, =\}$$

denote the thirteen Allen relations. Examples of these relations are given in Figure 4. As the flow of time, we use the real numbers with the usual ordering. Let $\mathsf{Int}_{\mathbb{R}}$ denote the set of all closed intervals $[r_1, r_2]$ over $\mathbb{R}$ with $r_1 < r_2$, i.e., point-intervals are not admitted. The extension $\mathsf{r}^{\mathbb{R}}$ of each Allen relation $\mathsf{r}$ is a subset of $\mathsf{Int}_{\mathbb{R}} \times \mathsf{Int}_{\mathbb{R}}$. It is defined in terms of the relationships between endpoints in the obvious way, c.f. Figure 4. We define the constraint system

$$\mathsf{Allen}_{\mathbb{R}} = \langle \mathsf{Allen}, \mathfrak{M}_{\mathbb{R}} \rangle$$

by setting $\mathfrak{M}_{\mathbb{R}} := \{N_{\mathbb{R}}\}$, where $N_{\mathbb{R}}$ is defined by fixing a variable $v_i \in \mathsf{Var}$ for every $i \in \mathsf{Int}_{\mathbb{R}}$ and setting

$$N_{\mathbb{R}} := \{(v_i \; r \; v_j) \mid r \in \mathsf{Allen},\ i, j \in \mathsf{Int}_{\mathbb{R}} \text{ and } (i, j) \in r^{\mathbb{R}}\}.$$

We could also define the constraint system $\mathsf{Allen}_{\mathbb{Q}}$ based on the rationals rather than on the reals: this has no impact on the satisfiability of finite and infinite Allen-networks (which are countable by definition). If we use the natural numbers or the integers, this still holds for finite networks, but not for infinite ones: there are infinite Allen-networks that are satisfiable over the reals and rationals, but not over the natural number and integers.

### 6.2.3 Properties of Constraint Systems

We will use constraint systems as a concrete domain for description logics. To obtain sound and complete reasoning procedures for DLs with such concrete domains, we require that constraint systems satisfy certain properties. First, we need to ensure that satisfiable networks (satisfying some additional conditions) can be "patched" together to a joint network that is also satisfiable. This is ensured by the patchwork property.

**Definition 6.3** [Patchwork Property] Let $\mathcal{C} = \langle \mathsf{Rel}, \mathfrak{M} \rangle$ be a constraint system, and let $N, M$ be finite complete $\mathsf{Rel}$-networks such that, for the intersection parts

$$\begin{aligned} I_{N,M} &:= \{(x\ r\ y) \mid x, y \in V_N \cap V_M \text{ and } (x\ r\ y) \in N\} \\ I_{M,N} &:= \{(x\ r\ y) \mid x, y \in V_N \cap V_M \text{ and } (x\ r\ y) \in M\} \end{aligned}$$

we have $I_{N,M} = I_{M,N}$. Then the *composition* of $N$ and $M$ is defined as $N \cup M$. We say that $\mathcal{C}$ has the *patchwork property* if the following holds: if $N$ and $M$ are satisfiable then $N \cup M$ is satisfiable.

The patchwork property is similar to the property of constraint networks formulated by Balbiani in [BC02], where constraint networks are combined with linear temporal logic.

For using constraint systems with the DL tableau algorithm presented here, we must be sure that, even if we patch together an infinite number of satisfiable networks, the resulting (infinite) network is still satisfiable. This is guaranteed by the compactness property.

**Definition 6.4** [Compactness] Let $\mathcal{C} = \langle \mathsf{Rel}, \mathfrak{M} \rangle$ be a constraint system. If $N$ is a $\mathsf{Rel}$-network and $V \subseteq V_N$, we write $N|_V$ to denote the network $\{(x\,r\,y) \in N \mid x, y \in V\} \subseteq N$. Then $\mathcal{C}$ has the *compactness property* if the following holds: a $\mathsf{Rel}$-network $N$ with $V_N$ infinite is satisfiable in $\mathcal{C}$ if and only if, for every finite $V \subseteq V_N$, the network $N|_V$ is satisfiable in $\mathcal{C}$.

Finally, our tableau algorithm has to check satisfiability of certain $\mathcal{C}$-networks. Thus, we have to assume that $\mathcal{C}$-satisfiability is decidable. The properties of constraint systems we require are summarized in the following definition.

**Definition 6.5** [$\omega$-admissible] Let $\mathcal{C} = (\mathsf{Rel}, \mathfrak{M})$ be a constraint system. We say that $\mathcal{C}$ is *$\omega$-admissible* iff the following holds:

1. satisfiability of finite $\mathcal{C}$-networks is decidable;
2. $\mathcal{C}$ has the patchwork property (c.f. Definition 6.3);
3. $\mathcal{C}$ has the compactness property (c.f. Definition 6.4).

In the appendixes of [LM07], we prove that $\mathsf{RCC8}_{\mathbb{R}^2}$ and $\mathsf{Allen}_{\mathbb{R}}$ satisfy the patchwork property and the compactness property. Moreover, satisfiability of finite networks is NP-complete (and thus decidable) in both systems: this is proved in [VKvB90] for $\mathsf{Allen}_{\mathbb{R}}$ and in [RN99] for $\mathsf{RCC8}_{\mathbb{R}^2}$. Thus, $\mathsf{RCC8}_{\mathbb{R}^2}$ and $\mathsf{Allen}_{\mathbb{R}}$ are $\omega$-admissible.

## 6.3 Syntax and Semantics

We introduce the description logic $\mathcal{ALC}(\mathcal{C})$, a superlogic of the basic DL $\mathcal{ALC}$ introduced in Section 2, that allows to define concepts with reference to the constraint system $\mathcal{C}$. Different incarnations of $\mathcal{ALC}(\mathcal{C})$ are obtained by instantiating it with different constraint systems.

**Definition 6.6** [$\mathcal{ALC}(\mathcal{C})$ Syntax and Semantics] Let $\mathcal{C} = (\mathsf{Rel}, \mathfrak{M})$ be a constraint system, and let $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_{cF}}$ be mutually disjoint and countably infinite sets of *concept names*, *role names*, and *concrete features*. We assume that $\mathsf{N_R}$ is partitioned into two countably infinite subsets $\mathsf{N_{aF}}$ and $\mathsf{N_{rR}}$. The elements of $\mathsf{N_{aF}}$ are called *abstract features* and the elements of $\mathsf{N_{rR}}$ *standard roles*. A *path* of length $k+1$ with $k \geq 0$ is a sequence $R_1 \cdots R_k g$ consisting of roles $R_1, \ldots, R_k \in \mathsf{N_R}$ and a concrete feature $g \in \mathsf{N_{cF}}$. A path $R_1 \cdots R_k g$ with $\{R_1, \ldots, R_k\} \subseteq \mathsf{N_{aF}}$ is called *feature path*. The set of $\mathcal{ALC}(\mathcal{C})$-concepts is the smallest set such that

1. every concept name $A \in \mathsf{N_C}$ is a concept,
2. if $C$ and $D$ are concepts and $R \in \mathsf{N_R}$, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, and $\exists R.C$ are concepts;
3. if $u_1$ and $u_2$ are feature paths and $r_1, \ldots, r_k \in \mathsf{Rel}$, then the following are also concepts:
$$\exists u_1, u_2.(r_1 \vee \cdots \vee r_k) \text{ and } \forall u_1, u_2.(r_1 \vee \cdots \vee r_k);$$
4. if $U_1$ and $U_2$ are paths of length at most two and $r_1, \ldots, r_k \in \mathsf{Rel}$, then the following are also concepts:
$$\exists U_1, U_2.(r_1 \vee \cdots \vee r_k) \text{ and } \forall U_1, U_2.(r_1 \vee \cdots \vee r_k);$$

Observe that we restrict the length of paths inside the constraint-based constructor to two only if at least one of the paths contains a standard role.

Interpretations for $\mathcal{ALC}(\mathcal{C})$, relative to those defined in Section 2, have as an additional argument a network $M$ from the set of models $\mathfrak{M}$ of $\mathcal{C}$. Thus, an *interpretation* $\mathcal{I}$ is a tuple $(\Delta_\mathcal{I}, \cdot^\mathcal{I}, M_\mathcal{I})$, where $\Delta_\mathcal{I}$ is a set called the *domain*, $\cdot^\mathcal{I}$ is the *interpretation function*, and $M_\mathcal{I} \in \mathfrak{M}$. The interpretation function maps concept and role names as defined in Section 2, and, additionally:

- each abstract feature $f$ to a partial function $f^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{I}$;
- each concrete feature $g$ to a partial function $g^\mathcal{I}$ from $\Delta_\mathcal{I}$ to the set of variables $V_{M_\mathcal{I}}$ of $M_\mathcal{I}$.

If $\mathbf{r} = r_1 \vee \cdots \vee r_k$, where $r_1, \ldots, r_k \in \mathsf{Rel}$, we write $M_\mathcal{I} \models (x\ \mathbf{r}\ y)$ iff there exists an $i \in \{1, \ldots, k\}$ such that $(x\ r_i\ y) \in M_\mathcal{I}$. The interpretation function is then extended to

the concepts of the form $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$ and $\forall R.C$ as defined in Section 2, and additionally:

$$\begin{aligned}(\exists U_1, U_2.\mathbf{r})^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \exists v_1 \in U_1^{\mathcal{I}}(d) \text{ and } v_2 \in U_2^{\mathcal{I}}(d) \\ &\qquad \text{with } M_{\mathcal{I}} \models (v_1\ \mathbf{r}\ v_2)\}, \\ (\forall U_1, U_2.\mathbf{r})^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \forall v_1 \in U_1^{\mathcal{I}}(d) \text{ and } v_2 \in U_2^{\mathcal{I}}(d), \\ &\qquad \text{we have } M_{\mathcal{I}} \models (v_1\ \mathbf{r}\ v_2)\}\end{aligned}$$

where for a path $U = R_1 \cdots R_k g$ and $d \in \Delta_{\mathcal{I}}$, $U^{\mathcal{I}}(d)$ is defined as

$$\begin{aligned}\{v \in V_{M_{\mathcal{I}}} \mid &\exists e_1, \ldots, e_{k+1} : d = e_1, \\ &(e_i, e_{i+1}) \in R_i^{\mathcal{I}} \text{ for } 1 \leq i \leq k, \text{ and } g^{\mathcal{I}}(e_{k+1}) = v\}.\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* of a concept $C$ iff $C^{\mathcal{I}} \neq \emptyset$. $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ iff it satisfies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all concept inclusions $C \sqsubseteq D$ in $\mathcal{T}$.

$C$ is called *satisfiable with respect to a TBox* $\mathcal{T}$ iff there exists a model of $C$ and $\mathcal{T}$. A concept $D$ *subsumes* a concept $C$ with respect to $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each model $\mathcal{I}$ of $\mathcal{T}$.

Observe that the network $M_{\mathcal{I}}$ in the previous definitionis a model of the constraint system $\mathcal{C}$, whence variables in this network correspond to values in $\mathcal{C}$ and are denoted with $v, v'$ rather than $x, y$.

The following example TBox describes some properties of a hotel using the constraint system $\mathsf{RCC8}_{\mathbb{R}^2}$, where has-room is a role, has-reception and has-carpark are abstract features (assuming that a hotel has at most a single reception and car park), loc is a concrete feature, and all capitalized words are concept names.

$$\begin{aligned}\mathsf{Hotel} &\sqsubseteq \forall \mathsf{has\text{-}room.Room} \sqcap \forall \mathsf{has\text{-}reception.Reception} \\ &\quad \sqcap \forall \mathsf{has\text{-}carpark.CarPark} \\ \mathsf{Hotel} &\sqsubseteq \forall(\mathsf{has\text{-}room\ loc}), (\mathsf{loc}).\mathsf{tpp} \vee \mathsf{ntpp} \\ &\quad \sqcap \forall(\mathsf{has\text{-}room\ loc}), (\mathsf{has\text{-}room\ loc}).\mathsf{dc} \vee \mathsf{ec} \vee \mathsf{eq} \\ \mathsf{CarFriendlyHotel} &\doteq \mathsf{Hotel} \sqcap \exists(\mathsf{has\text{-}reception\ loc}), (\mathsf{loc}).\mathsf{tpp} \\ &\quad \sqcap \exists(\mathsf{has\text{-}carpark\ loc}), (\mathsf{loc}).\mathsf{ec} \\ &\quad \sqcap \exists(\mathsf{has\text{-}carpark\ loc}), (\mathsf{has\text{-}reception\ loc}).\mathsf{ec}\end{aligned}$$

The first concept inclusion expresses that hotels are related via the three roles to objects of the proper type. The second concept inclusion says that the rooms of a hotel are spatially contained in the hotel, and that rooms do not overlap. Finally, the last concept inclusion describes hotels that are convenient for car owners: they have a carpark that is directly next to the reception. This situation is illustrated in Figure 5.

The most important reasoning tasks for DLs are satisfiability and subsumption: a concept $C$ is called *satisfiable with respect to a TBox* $\mathcal{T}$ iff there exists a common model of $C$ and $\mathcal{T}$. A concept $D$ *subsumes* a concept $C$ with respect to $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each model $\mathcal{I}$ of $\mathcal{T}$. It is well-known that subsumption can be reduced to (un)satisfiability: $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$. This allows us to concentrate on concept satisfiability when devising reasoning procedures.

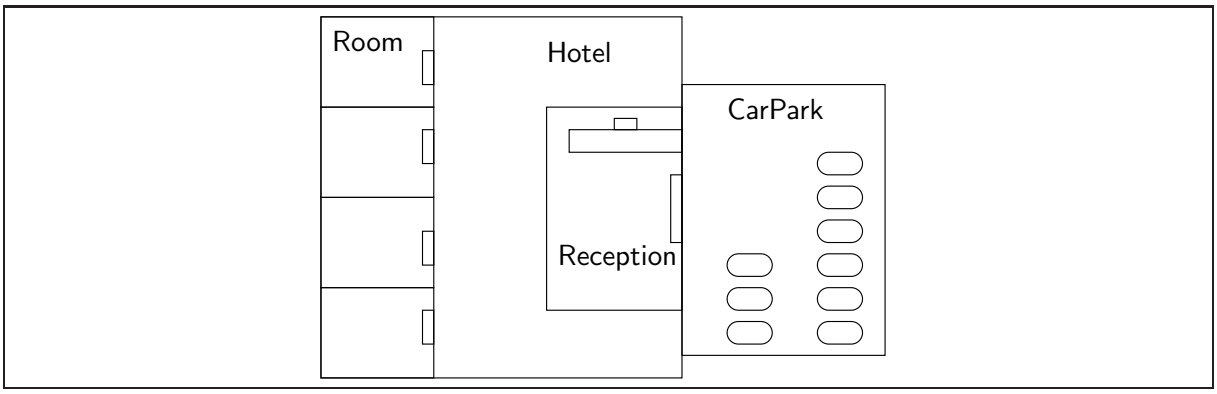


Figure 5: An example of a CarFriendlyHotel.

## 6.4 A Tableau Algorithm for $\mathcal{ALC}(\mathcal{C})$

We present a tableau algorithm that decides satisfiability of $\mathcal{ALC}(\mathcal{C})$-concepts w.r.t. TBoxes. Tableau algorithms are among the most popular decision procedures for description logics since they are amenable to various optimization techniques and often can be efficiently implemented. Therefore, we view the algorithm presented here as a first step towards practicable reasoning with concrete domains and general TBoxes. On the flipslide, algorithms such as the one developed in this section usually do not yield tight upper complexity bounds.

The algorithm developed in the following is independent of the constraint system $\mathcal{C}$. This is achieved by delegating reasoning in $\mathcal{C}$ to an external reasoner that decides satisfiability of $\mathcal{C}$-networks. Throughout this section, we assume $\mathcal{C}$ to be $\omega$-admissible.

### 6.4.1 Normal Forms

It is convenient to first convert the input concept and TBox into an appropriate syntactic form. More precisely, we convert concepts and TBoxes into negation normal form (NNF) and restrict the length of paths that appear inside the constraint-based concept constructors. We start with describing NNF conversion. A concept is said to be in *negation normal form* if negation occurs only in front of concept names. The following lemma shows that NNF can be assumed without loss of generality. For a path $U = R_1 \cdots R_k g$, we write $\mathsf{ud}(U)$ to denote the concept $\forall R_1. \cdots \forall R_k.(\forall g, g.r \sqcap \forall g, g.r')$ where $r, r' \in \mathsf{Rel}$ are arbitrary such that $r \neq r'$.[4]

**Lemma 6.7 (NNF Conversion)** *Exhaustive application of the following rewrite rules translates $\mathcal{ALC}(\mathcal{C})$-concepts to equivalent ones in NNF.*

$$\begin{array}{rcll} \neg\neg C & \rightsquigarrow & C & \\ \neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D \\ \neg(\exists R.C) & \rightsquigarrow & (\forall R.\neg C) & \neg(\forall R.C) \rightsquigarrow (\exists R.\neg C) \end{array}$$

[4]This presupposes the natural assumptions that Rel has cardinality at least two.

$$\neg(\forall U_1, U_2.(r_1 \vee \cdots \vee r_k)) \rightsquigarrow \begin{cases} \bot & \textit{if } \mathsf{Rel} = \{r_1, \ldots, r_k\} \\ \exists U_1, U_2.(\bigvee_{r \in \mathsf{Rel} \setminus \{r_1, \ldots, r_k\}} r) & \textit{otherwise} \end{cases}$$

$$\neg(\exists U_1, U_2.(r_1 \vee \cdots \vee r_k)) \rightsquigarrow \begin{cases} \mathsf{ud}(U_1) \sqcup \mathsf{ud}(U_2) & \textit{if } \mathsf{Rel} = \{r_1, \ldots, r_k\} \\ \forall U_1, U_2.(\bigvee_{r \in \mathsf{Rel} \setminus \{r_1, \ldots, r_k\}} r) & \textit{otherwise} \end{cases}$$

*By* $\mathsf{nnf}(C)$*, we denote the result of converting* $C$ *into NNF using the above rules.*

In Lemma 6.7, the last two transformations are equivalence preserving since the Rel-networks used as models in $\mathcal{C}$ are complete.

We now show how to restrict the length of paths by converting concepts and TBoxes into path normal form. This normal form was first considered in [Lut04a] in the context of the description logic $\mathcal{TDL}$ and in [Lut02a] in the context of $\mathbb{Q}$-$\mathcal{SHIQ}$.

**Definition 6.8** [Path Normal Form] An $\mathcal{ALC}(\mathcal{C})$-concept $C$ is in *path normal form (PNF)* if it is in NNF and for all subconcepts

$$\exists U_1, U_2.(r_1 \vee \cdots \vee r_k) \text{ and } \forall U_1, U_2.(r_1 \vee \cdots \vee r_k)$$

of $C$, the length of $U_1$ and $U_2$ is at most two. An $\mathcal{ALC}(\mathcal{C})$-TBox $\mathcal{T}$ is in path normal form iff $\mathcal{T}$ is of the form $\{\top \sqsubseteq C\}$, with $C$ in PNF.

The following lemma shows that we can w.l.o.g. assume $\mathcal{ALC}(\mathcal{C})$-concepts and TBoxes to be in PNF.

**Lemma 6.9** *Satisfiability of* $\mathcal{ALC}(\mathcal{C})$*-concepts w.r.t. TBoxes can be reduced in polynomial time to satisfiability of* $\mathcal{ALC}(\mathcal{C})$*-concepts in PNF w.r.t. TBoxes in PNF.*

The previous lemma shows that, in what follows, we may assume w.l.o.g. that all concepts and TBoxes are in PNF.

### 6.4.2 Data Structures

We introduce the data structures underlying the tableau algorithm, an operation for extending this data structure, and a cycle detection mechanism that is needed to ensure termination of the algorithm. As already said, we assume that the input concept $C_0$ is in PNF, and that the input TBox $\mathcal{T}$ is of the form $\mathcal{T} = \{\top \sqsubseteq C_\mathcal{T}\}$, where $C_\mathcal{T}$ is in PNF.

The main ingredient of the data structure underlying our algorithm is a tree that, in case of a successful run of the algorithm, represents a single model of the input concept and TBox. Due to the presence of the constraint system $\mathcal{C}$, this tree has two types of nodes: abstract ones that represent individuals of the logic domain $\Delta_\mathcal{I}$ and concrete ones that represent values of the concrete domain. We use $\mathsf{sub}(C)$ to denote the set of subconcepts of the concept $C$ and set $\mathsf{sub}(C_0, \mathcal{T}) := \mathsf{sub}(C_0) \cup \mathsf{sub}(C_\mathcal{T})$.

**Definition 6.10** [Completion system] Let $\mathsf{O_a}$ and $\mathsf{O_c}$ be disjoint and countably infinite sets of *abstract nodes* and *concrete nodes*. A *completion tree* for an $\mathcal{ALC}(\mathcal{C})$-concept $C$ and a TBox $\mathcal{T}$ is a finite, labeled tree $T = (\mathsf{V_a}, \mathsf{V_c}, E, \mathcal{L})$ with nodes $\mathsf{V_a} \cup \mathsf{V_c}$ and edges $E \subseteq (\mathsf{V_a} \times (\mathsf{V_a} \cup \mathsf{V_c}))$ such that $\mathsf{V_a} \subseteq \mathsf{O_a}$ and $\mathsf{V_c} \subseteq \mathsf{O_c}$. The tree is labeled as follows:

1. each node $a \in \mathsf{V_a}$ is labeled with a subset $\mathcal{L}(a)$ of $\mathsf{sub}(C, \mathcal{T})$,
2. each edge $(a, b) \in E$ with $a, b \in \mathsf{V_a}$ is labeled with a role name $\mathcal{L}(a, b)$ occurring in $C$ or $\mathcal{T}$;
3. each edge $(a, x) \in E$ with $a \in \mathsf{V_a}$ and $x \in \mathsf{V_c}$ is labeled with a concrete feature $\mathcal{L}(a, x)$ occurring in $C$ or $\mathcal{T}$.

A node $b \in \mathsf{V_a}$ is an *R-successor* of a node $a \in \mathsf{V_a}$ if $(a, b) \in E$ and $\mathcal{L}(a, b) = R$, while $x \in \mathsf{V_c}$ is a *g-successor* of $a$ if $(a, x) \in E$ and $\mathcal{L}(a, x) = g$. The notion *U-successor* for a path $U$ is defined in the obvious way.

A *completion system* for an $\mathcal{ALC}(\mathcal{C})$-concept $C$ and a TBox $\mathcal{T}$ is a pair $S = (T, \mathcal{N})$ where $T = (\mathsf{V_a}, \mathsf{V_c}, E, \mathcal{L})$ is a completion tree for $C$ and $\mathcal{T}$ and $\mathcal{N}$ is a $\mathsf{Rel}$-network with $V_{\mathcal{N}} = \mathsf{V_c}$.

We now define an operation that is used by the tableau algorithm to add new nodes to completion trees. The operation respects the functionality of abstract and concrete features.

**Definition 6.11** [$\oplus$ Operation] An abstract or concrete node is called *fresh* in a completion tree $T$ if it does not appear in $T$. Let $S = (T, \mathcal{N})$ be a completion system with $T = (\mathsf{V_a}, \mathsf{V_c}, E, \mathcal{L})$. We use the following operations:

- if $a \in \mathsf{V_a}$, $b \in \mathsf{O_a}$ fresh in $T$, and $R \in \mathsf{N_R}$, then $S \oplus aRb$ yields the completion system obtained from $S$ in the following way:
  - if $R \notin \mathsf{N_{aF}}$ or $R \in \mathsf{N_{aF}}$ and $a$ has no $R$-successors, then add $b$ to $\mathsf{V_a}$, $(a, b)$ to $E$ and set $\mathcal{L}(a, b) = R$, $\mathcal{L}(b) = \emptyset$.
  - if $R \in \mathsf{N_{aF}}$ and there is a $c \in \mathsf{V_a}$ such that $(a, c) \in E$ and $\mathcal{L}(a, c) = R$ then rename $c$ in $T$ with $b$.
- if $a \in \mathsf{V_a}$, $x \in \mathsf{O_c}$ fresh in $T$, and $g \in \mathsf{N_{cF}}$, then $S \oplus agx$ yields the completion system obtained from $S$ in the following way:
  - if $a$ has no $g$-successors, then add $x$ to $\mathsf{V_c}$, $(a, x)$ to $E$ and set $\mathcal{L}(a, x) = g$;
  - if $a$ has a $g$-successor $y$, then rename $y$ in $T$ and $\mathcal{N}$ with $x$.

Let $U = R_1 \cdots R_n g$ be a path. With $S \oplus aUx$, where $a \in \mathsf{V_a}$ and $x \in \mathsf{O_c}$ is fresh in $T$, we denote the completion system obtained from $S$ by taking distinct nodes $b_1, ..., b_n \in \mathsf{O_a}$ which are fresh w.r.t. $T$ and setting

$$S' := S \oplus aR_1b_1 \oplus \cdots \oplus b_{n-1}R_nb_n \oplus b_ngx$$

The tableau algorithm works by starting with an initial completion system that is then successively expanded with the goal of constructing a model of the input concept and TBox. To ensure termination, we need a mechanism for detecting cyclic expansions, which is commonly called *blocking*. Informally, we detect nodes in the completion tree that are

similar to previously created ones and then block them, i.e., stop further expansion at such nodes. To introduce blocking, we start with some preliminaries. For $a \in \mathsf{V_a}$, we define the set of features of $a$ as

$$\mathsf{feat}(a) := \{\ g \in \mathsf{N_{cF}} \mid a \text{ has a } g\text{-successor}\ \}.$$

Next, we define the *concrete neighborhood* of $a$ as the constraint network

$$\begin{array}{rcl} \mathcal{N}(a) & := & \{\ (x\ r\ y) \mid \text{ there exist } g, g' \in \mathsf{feat}(a) \text{ s.t. } x \text{ is a } g\text{-succ.} \\ & & \qquad\qquad \text{of } a,\ y \text{ is a } g'\text{-succ. of } a, \text{ and } (x\ r\ y) \in \mathcal{N}\ \} \end{array}$$

Finally, if $a, b \in \mathsf{V_a}$ and $\mathsf{feat}(a) = \mathsf{feat}(b)$, we write $\mathcal{N}(a) \sim \mathcal{N}(b)$ to express that $\mathcal{N}(a)$ and $\mathcal{N}(b)$ are isomorphic, i.e., that the mapping $\pi : V_{\mathcal{N}(a)} \to V_{\mathcal{N}(b)}$ defined by mapping the $g$-successor of $a$ to the $g$-successor of $b$ for all $g \in \mathsf{feat}(a)$ is an isomorphism.

If $T$ is a completion tree and $a$ and $b$ are abstract nodes in $T$, then we say that $a$ is an *ancestor* of $b$ if $b$ is reachable from $a$ in the tree $T$.

**Definition 6.12** [Blocking] Let $S = (T, \mathcal{N})$ be a completion system for a concept $C_0$ and a TBox $\mathcal{T}$ with $T = (\mathsf{V_a}, \mathsf{V_c}, E, \mathcal{L})$, and let $a, b \in \mathsf{V_a}$. We say that $a \in \mathsf{V_a}$ is *potentially blocked by* $b$ if the following holds:

1. $b$ is an ancestor of $a$ in T,
2. $\mathcal{L}(a) \subseteq \mathcal{L}(b)$,
3. $\mathsf{feat}(a) = \mathsf{feat}(b)$.

We say that $a$ is *directly blocked* by $b$ if the following holds:

1. $a$ is potentially blocked by $b$,
2. $\mathcal{N}(a)$ and $\mathcal{N}(b)$ are complete, and
3. $\mathcal{N}(a) \sim \mathcal{N}(b)$.

Finally, $a$ is *blocked* if it or one of its ancestors is directly blocked.

### 6.4.3 The Tableau Algorithm

To decide the satisfiability of an $\mathcal{ALC}(\mathcal{C})$-concept $C_0$ w.r.t. a TBox $\mathcal{T}$, the tableau algorithm is started with the initial completion system $S_{C_0} = (T_{C_0}, \emptyset)$, where the initial completion tree $T_{C_0}$ is defined by setting

$$T_{C_0} := (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C_0\}\}).$$

The algorithm then repeatedly applies the completion rules given in Figure 6. In the formulation of $R$net, a *completion* of a Rel-network $N$ is a satisfiable and complete Rel-network $N'$ such that $V_N = V_{N'}$ and $N \subseteq N'$. Later on, we will argue that the completion to be guessed always exists.

$R\sqcap$ if $C_1 \sqcap C_2 \in \mathcal{L}(a)$, $a$ is not blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(a)$,
then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_1, C_2\}$

$R\sqcup$ if $C_1 \sqcup C_2 \in \mathcal{L}(a)$, $a$ is not blocked, and $\{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset$,
then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$R\exists$ if $\exists R.C \in \mathcal{L}(a)$, $a$ is not blocked, and there is no $R$-successor $b$ of $a$ such that $C \in \mathcal{L}(b)$
then set $S := S \oplus aRb$ for a fresh $b \in O_a$ and $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$

$R\forall$ if $\forall R.C \in \mathcal{L}(a)$, $a$ is not blocked, and $b$ is an $R$-successor of $a$ such that $C \notin \mathcal{L}(b)$
then set $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$

$R\exists_c$ if $\exists U_1, U_2.(r_1 \vee \cdots \vee r_k) \in \mathcal{L}(a)$, $a$ is not blocked, and there exist no $x_1, x_2 \in V_c$ such that $x_i$ is a $U_i$-successor of $a$ for $i = 1, 2$ and $(x_1\ r_i\ x_2) \in \mathcal{N}$ for some $i$ with $1 \leq i \leq k$
then set $S := S \oplus aU_1x_1 \oplus aU_2x_2$ with $x_1, x_2 \in \mathsf{O_c}$ fresh and $\mathcal{N} := \mathcal{N} \cup \{(x_1\ r_i\ x_2)\}$ for some $i$ with $1 \leq i \leq k$

$R\forall_c$ if $\forall U_1, U_2.(r_1 \vee \cdots \vee r_k) \in \mathcal{L}(a)$, $a$ is not blocked, and there are $x_1, x_2 \in \mathsf{V_c}$ such that $x_i$ is a $U_i$-successor of $a$ for $i = 1, 2$ and $(x_1\ r_i\ x_2) \notin \mathcal{N}$ for all $i$ with $1 \leq i \leq k$
then set $\mathcal{N} := \mathcal{N} \cup \{(x_1\ r_i\ x_2)\}$ for some $i$ with $1 \leq i \leq k$

$R\mathsf{net}$ if $a$ is potentially blocked by $b$ or vice versa and $\mathcal{N}(a)$ is not complete
then non-deterministically guess a completion $\mathcal{N}'$ of $\mathcal{N}(a)$ and set $\mathcal{N} := \mathcal{N} \cup \mathcal{N}'$

$R\mathsf{tbox}$ if $C_{\mathcal{T}} \notin \mathcal{L}(a)$
then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_{\mathcal{T}}\}$

Figure 6: The completion rules.

As has already been noted above, rule application can be understood as the step-wise construction of a model of $C_0$ and $\mathcal{T}$. Among the rules, there are four non-deterministic ones: $R\sqcup$, $R\exists_c$, $R\forall_c$, and $R\mathsf{net}$.[5] Rules are applied until an obvious inconsistency (as defined below) is detected or the completion system becomes *complete*, i.e., no more rules are applicable. The algorithm returns "satisfiable" if there is a way to apply the rules such that a complete completion system is found that does not contain a contradiction. Otherwise, it returns "unsatisfiable".

All rules except $R\mathsf{net}$ are rather standard, see for example [BH91, Lut02b].[6] The purpose of $R\mathsf{net}$ is to resolve a potential blocking situation between two nodes $a$ and $b$ into either an actual blocking situation or a non-blocking situation. This is achieved by completing the networks $\mathcal{N}(a)$ and $\mathcal{N}(b)$. For ensuring termination, an appropriate

[5] By disallowing disjunctions of relations in the constraint-based concept constructors, $R\exists_c$ and $R\forall_c$ can easily be made deterministic.

[6] Note that our version of the $R\exists$ rule uses the operation $S \oplus aRb$ which initializes the label $\mathcal{L}(b)$, and thus the rule only *adds* $C$ to the already existing label.

```
procedure sat(S)
 if S contains a clash then return unsatisfiable
 if S is complete then return satisfiable
 if Rnet is applicable
  then S' := application of Rnet to S
  else S' := application of any applicable completion rule to S
 return sat(S')
```

Figure 7: The (non-deterministic) algorithm for satisfiability in $\mathcal{ALC}(\mathcal{C})$.

interplay between this rule and the blocking condition is crucial. Namely, we have to apply $R\mathsf{net}$ with highest precedence. It can be seen that the blocking mechanism obtained in this way is a refinement of pairwise blocking as known from [HST99]. In particular, the conditions $\mathcal{L}(a) \subseteq \mathcal{L}(b)$ and $\mathsf{feat}(a) = \mathsf{feat}(b)$ are implied by the standard definition of pairwise blocking due to path normal form.

We now define what we mean by an obvious inconsistency. As soon as such an inconsistency is encountered, the tableau algorithm returns "unsatisfiable".

**Definition 6.13** [Clash] Let $S = (T, \mathcal{N})$ be a completion system for a concept $C$ and a TBox $\mathcal{T}$ with $T = (\mathsf{V_a}, \mathsf{V_c}, E, \mathcal{L})$. $S$ contains a *clash* if one of the following conditions holds:

1. there is an $a \in \mathsf{V_a}$ and an $A \in \mathsf{N_C}$ such that $\{A, \neg A\} \subseteq \mathcal{L}(a)$;
2. $\mathcal{N}$ is not satisfiable in $\mathcal{C}$.

If $S$ does not contain a clash, $S$ is called *clash-free*.

We present the tableau algorithm in pseudo-code notation in Figure 7. It is started with the initial completion system as argument, i.e., by calling $\mathsf{sat}(S_{C_0})$.

Note that checking for clashes before rule application is crucial for $R\mathsf{net}$ to be well-defined: if $R\mathsf{net}$ is applied to a node $a$, we must be sure that there indeed exists a completion $\mathcal{N}'$ of $\mathcal{N}(a)$ to be guessed, i.e., a *satisfiable* network $\mathcal{N}'$ such that $V'_{\mathcal{N}} = V_{\mathcal{N}(a)}$ and $\mathcal{N}(a) \subseteq \mathcal{N}'$. Clash checking before rule application ensures that the network $\mathcal{N}$ is satisfiable when $R\mathsf{net}$ is applied. Clearly, this implies the existence of the required completion.

Termination, soundness and completeness of the presented tableau algorithm are proved in [LM07]. Thus, we get the following theorem:

**Theorem 6.14** *If $\mathcal{C}$ is an $\omega$-admissible constraint system, the tableau algorithm decides satisfiability of $\mathcal{ALC}(\mathcal{C})$ concepts w.r.t. general TBoxes.*

A close inspection of our algorithm shows that it runs in 2-NExpTime if $\mathcal{C}$-satisfiability is in NP. We conjecture that, by mixing the techniques presented here with those from [Lut04a, Lut02a], it is possible to prove ExpTime-completeness of satisfiability in $\mathcal{ALC}(\mathcal{C})$ provided that satisfiability in $\mathcal{C}$ can be decided in ExpTime. Various language extensions such as transitive roles and number restrictions should also be possible in a straightforward way.

## 6.5 Practicability

With Theorem 6.14, we have achieved the our main aim: providing a general decidability result for description logics with both general TBoxes and concrete domains. Our second aim is to identify an algorithm that is more practicable than the existing approaches based on automata [Lut04a, Lut02a], i.e., that can be implemented such that an acceptable runtime behaviour is observed on realistic inputs. Since we have not yet implemented our algorithm,[7] an empirical evaluation is out of reach. In the following, we discuss the practicability on a general level.

Regarding an efficient implementation, the main difficulties of our algorithm compared with successfully implemented tableau algorithms such as the ones in [SS91, HS99] are the following:

- Our algorithm requires satisfiability checks of the network $\mathcal{N}$ constructed as part of the completion system. The problem is that this check involves the *whole* network $\mathcal{N}$ rather than only small parts of it. In practice, the constructed completion systems (and associated networks) are often too large to be considered as a whole.
- The rules $R\exists_c$, $R\forall_c$, and $R\mathsf{net}$ introduce additional non-determinism. In implementations, this non-determinism induces backtracking.

It is possible that these difficulties can be overcome by developing appropriate heuristics and optimization techniques. However, there is also an easy way around them. In the following, we argue that there is a fragment of our language that still provides interesting expressive power and in which the implementation difficulties discussed above are non-existent.

The fragment of $\mathcal{ALC}(\mathcal{C})$ that we consider is obtained by making the following assumptions:

- There is only a single concrete feature $g$. Note that this is acceptable with constraint systems such as $\mathsf{RCC8}_{\mathbb{R}^2}$ and $\mathsf{Allen}_{\mathbb{R}}$, where $g$ could be $\mathsf{has\text{-}extension}$ and $\mathsf{has\text{-}lifetime}$, respectively.
- There are no paths of length greater than 2, i.e., Clause 3 is eliminated from Definition 6.6. This is necessary since we need to introduce additional concrete features to establish path normal form if Clause 3 is present. We believe that paths of length three or more are only needed in exceptional cases, anyway.
- There exists a unique equality predicate $\mathsf{eq}$ in $\mathcal{C}$, i.e., for all models $N \in \mathfrak{M}$ and all $v \in V_N$, we have $(v\ \mathsf{eq}\ v) \in N$.

Going to this fragment of $\mathcal{ALC}(\mathcal{C})$ allows the following simplification of our tableau algorithm.

1. The non-deterministic $R\mathsf{net}$ rule can simply be dropped because, for each abstract node $a$, the network $\mathcal{N}(a)$ is either empty or consists of a single node that is related to itself via $\mathsf{eq}$. Thus, every potential blocking situation is an actual blocking situation.

[7] This is a non-trivial task since a large number of sophisticated optimization techniques is required, c.f. [HPS99].

2. We can localize the satisfiability check of the network $\mathcal{N}$ as follows. For $a \in \mathsf{V_a}$, let $\widehat{\mathcal{N}}(a)$ denote the restriction of $\mathcal{N}$ to the $g$-successor of $a$ and the $g$-successors of all abstract successors of $a$. Instead of checking the whole network $\mathcal{N}$ for satisfiability, we separately check, for each $a \in \mathsf{V_a}$, satisfiability of $\widehat{\mathcal{N}}(a)$. It can be seen as follows that this is equivalent to a global check: first, $\mathcal{C}$ has the patchwork property. Second, due to the fact that there is only a single concrete feature $g$, the networks $\widehat{\mathcal{N}}(a)$ overlap at single nodes only. Due to the presence of the equality predicate $\mathsf{eq}$, the overlapping part of two such networks is thus complete. Finally, it is easy to see that the patchwork property implies a more general version of itself where only the overlapping part of the two involved networks is complete, but the networks themselves are not.

Hence, the only difficulty that remains is the non-determinism of the rules $R\exists_c$ and $R\forall_c$. However, we believe that this non-determinism is not too difficult to deal with. To see this, observe that the non-deterministic choices made by these rules have only a very local impact: they only influence the outcome of the satisfiability check of the relevant local network $\widehat{\mathcal{N}}(a)$. Therefore, it does not seem necessary to implement a complex backtracking/backjumping machinery. If the concrete domain reasoner used for deciding $\mathcal{C}$-satisfiability supports disjunctions, it is even possible to push the non-determinism out of the tableau algorithm into the reasoner for $\mathcal{C}$-satisfiability. Roughly, one would need to allow disjunctions in the constraint network $\mathcal{N}$ and pass these on to the reasoner for $\mathcal{C}$.

## 6.6 Related Work

Concrete domains in the context of DL were first introduced in [BH91], where the first tableau algorithm for deciding consistency of $\mathcal{ALC}(\mathcal{D})$-ABoxes is developed and shown that reasoning in $\mathcal{ALC}(\mathcal{D})$ is PSPACE-complete.

Various extensions of $\mathcal{ALC}(\mathcal{D})$ (with acyclic TBoxes, feature agreements, role forming operators etc.) were treated in [HLM98, Lut01, Lut04b, Lut02b]. Even seemingly harmless extensions were shown to lead to NEXPTIME-complete reasoning and some of them even to undecidability. Most importantly, it was shown that reasoning in the presence of general TBoxes is undecidable for a large class of concrete domains [Lut04b]. For a survey on DLs with concrete domains see [Lut03].

In [Lut04a, Lut02a] it is shown that reasoning in the DLs $\mathcal{ALC}$ and $\mathcal{SHIQ}$ extended with a temporal concrete domain and a concrete domain based on rationals and relations $\leq$, $=_5$,..., respectively, and general TBoxes is decidable and EXPTIME-complete.

A reasoning procedure for the epressive DL with a concrete domain $\mathcal{SHOQ}(\mathcal{D})$, which is underlying the web ontology language OWL, was presented in [HS01]. DLs with concrete domains and different kinds of key constraints were thoroughly investigated in [LAHS05, LM04, DK06].

The DL reasoner RACER supports concrete domains based on real numbers where the paths appearing in concepts are restricted to concrete features [HM01, HMW01].

# 7 Pinpointing in Expressive DLs

## 7.1 Introduction

The main advantage of reasoning support during ontology design is that modelling mistakes can be automatically detected and reported to the user. However, simply reporting a mistake is often not enough since it does not help the designer in understanding and resolving the problem. Therefore, error management comprises much more than only reporting a problem. Additionally, the source of the problem has to be pinpointed and the problem can be explained to the designer in detail in an understandable way. Going even further, a reasoner can make suggestions on how to resolve the problem.

Error management is as important to ontology design and maintenance as debugging is to software engineering. As discussed in previous sections, the most important errors that can occur during ontology design and automatically detected by reasoners are the following:

- *Inconsistent Ontology.* The ontology is contradictory in itself. Logically, anything follows from such an ontology. Therefore, a contradictory ontology cannot be used in practice before the inconsistency is resolved.
- *Inconsistent Concept Description.* Though less severe than an inconsistent ontology, an inconsistent concept description also indicates modelling flaws in the ontology: inconsistent concept description cannot have any instances, and thus does not represent any realistic concept in any domain.
- *Unintended Subsumptions.* Even though the ontology is consistent and does not involve any inconsistent concept descriptions, it might still contain errors. Notably, there can be a surplus subsumptions, i.e. subsumption relationships that are implied by the ontology but neither desired nor expected.
- *Missing Subsumption.* The opposite of the previous, where a subsumption relationship is expected to follow from the ontology, but does not. Can be fixed by explicitly adding the missing subsumption.

Available ontology reasoners are able to determine the existence of the above errors. Unfortunately, they usually only report the existence of an error, without providing any further information. If the designed ontology is large and there are complex interactions between the defined concepts, it may be very difficult to find the source of the problem. For example, a single inconsistent concept description may result in hundreds of other concept descriptions becoming inconsistent as well. If the ontology designer is only presented with a list of hundreds of unsatisfiable concepts, it is far from clear where to look for locating the problem. Even if the problem is located, it may be difficult to understand it and it may be unclear how to resolve it.

In order to support error management, the reasoning backend must first be able to pinpoint to the cause of error what causes it. The answer to this question can be, for example, in the form of a minimal set of concept descriptions responsible for such an error. Also, after explanation of the error, an unexperienced developer may not know what

actions to take to resolve the errors. To alleviate this problem, the reasoning backend can suggest how to resolve the problem making as little change as possible. Such a suggestion for revision may be of the form that describes what measures to take to rectify the ontology. It might also display to the experts a number of possibilities to fix the error.

In description logic, various approaches to error pinpointing and explanation have been studied. For example, the computation of Minimal Unsatisfiability Preserving SubTBoxes (MUPS) can be used to pinpoint the source of the inconsistency of a concept description. In case of missing subsumptions, a small counter-model can be generated as explanations. In case of too many subsumptions, the derivation steps of some proof calculus can be used as a basis to obtain explanation. When the error is to be automatically resolved, methods from belief revision become relevant.

In this section, we present a set of algorithms for axiom pinpointing, in particular for finding all the justifications for an entailment. The algorithms come in two flavors:

1. *Reasoner dependent (or Glass-box)* algorithms are built on existing tableau-based decision procedures for expressive Description Logics. Their implementation requires a thorough and non-trivial modification of the internals of the reasoner.

2. *Reasoner independent (or Black-box)* algorithms use the DL reasoner solely as a *subroutine* and the internals of the reasoner do not need to be modified. The reasoner behaves as a "Black-box" that accepts, as usual, a concept and a KB as input and returns an affirmative or a negative answer, depending on whether the concept is satisfiable or not w.r.t. the KB. In order to obtain the justifications, the axiom pinpointing algorithm selects the appropriate inputs to the DL reasoner and interprets its output accordingly.

Glass-box algorithms typically affect many aspects of the internals of the reasoner and strongly depend on the DL under consideration.

Black-box algorithms typically require many satisfiability tests, but they can be easily and robustly implemented, since they only rely on the availability of a sound and complete reasoner for such a DL. Consequently, using a Black-box approach, the service can also be implemented on reasoners that are based on techniques other than tableaux, such as resolution.

We also investigate *hybrid* algorithms, which combine Glass-box and Black-box approaches to obtain sound and complete solutions relatively easily, i.e., without dealing with complicated implementation issues. The idea here is to use one of the approaches to reduce the problem space significantly and the other as a post-processing step to obtain the correct solution.

Finally, we propose a collection of techniques for *repairing* errors, once the the axioms responsible for the error have been identified.

## 7.2 Justification of Entailments and MUPS

In this section, we provide a formal definition of justifications and introduce the notion of a MUPS, as described in [SC03a]. Finally, we show how justifications and MUPS relate to each other for the description logic $\mathcal{SHOIN}$.

We start with the definition of justifications.

**Definition 7.1** [Justification] Let $\mathcal{T} \models \alpha$ where $\alpha$ is a sentence. A fragment $\mathcal{T}' \subseteq \mathcal{T}$ is a justification for $\alpha$ in $\mathcal{T}$ if $\mathcal{T}' \models \alpha$, and $\mathcal{T}'' \not\models \alpha$ for every $\mathcal{T}'' \subset \mathcal{T}'$.

We denote by $JUST(\alpha, \mathcal{T})$ the set of all the justifications for $\alpha$ in $\mathcal{T}$. Given $\alpha$ and $\mathcal{T}$, the *Axiom Pinpointing* inferential service is the problem of computing $JUST(\alpha, \mathcal{T})$

MUPS are formally defined as follows:

**Definition 7.2** [MUPS] Let $C$ be a concept, which is unsatisfiable w.r.t. a knowledge base $\mathcal{T}$. A fragment $\mathcal{T}' \subseteq \mathcal{T}$ is a MUPS of $C$ in $\mathcal{T}$ if $C$ is unsatisfiable in $\mathcal{T}'$, and $C$ is satisfiable in every $\mathcal{T}'' \subset \mathcal{T}'$.

We denote by $MUPS(C, \mathcal{T})$ the set of all the MUPS for $C$ in $\mathcal{T}$. When the KB we are referring to is clear from the context, we will relax the notation and use $MUPS(C)$ instead.

The relationship between MUPS and justifications is established by Proposition 7.3. The simple theorem is based on the following on the following well-known result: given a $\mathcal{SHOIN}$ knowledge base $\mathcal{T}$, for every sentence (axiom or assertion) $\alpha$ entailed by $\mathcal{T}$, there is always a concept $C_\alpha$ that is unsatisfiable w.r.t $\mathcal{T}$. Conversely, given any concept $C$ that is unsatisfiable w.r.t. $\mathcal{T}$, there is always a sentence $\alpha_C$ that is entailed by $\mathcal{T}$. Consequently, given a $\mathcal{SHOIN}$ KB, the problem of finding all the MUPS for an unsatisfiable concept and the problem of finding all the justifications for a given entailment can be reduced to each other.

**Proposition 7.3** *Let $\mathcal{T}$ be a knowledge base, $\alpha$ be a sentence and let $C_\alpha$ be a concept s.t. for every KB $\mathcal{T}' \subseteq \mathcal{T}$, $\mathcal{T}' \models \alpha \Leftrightarrow C_\alpha$ is unsatisfiable w.r.t. $\mathcal{T}'$, then $JUST(\alpha, \mathcal{T}) = MUPS(C_\alpha, \mathcal{T})$*

In the remainder of this section, we shall restrict our attention, without loss of generality, to the problem of finding all the MUPS for an unsatisfiable concept w.r.t to a $\mathcal{SHOIN}$ KB.

**Note:** The notion of justifications can be easily extended to include justifications for an *inconsistent* KB, i.e., minimal sets of axioms responsible for making a KB logically inconsistent. Also, all the ensuing algorithms for finding justifications for unsatisfiability entailments are directly applicable to finding justifications for inconsistency. This should be no surprise as unsatisfiability detection is performed by attempting to generate an inconsistent ontology.

## 7.3 Finding Justifications

In this section, we investigate the problem of computing justifications. First, we focus on the problem of finding *one* justification and then we discuss how to compute *all* of them. For such a purpose, we explore and discuss different techniques to tackle the problem.

### 7.3.1 Computing a Single Justification

The most common strategy for computing a single justification is to keep track of the axioms from the KB responsible for each change in the completion graph in a tableau expansion, namely, the addition of a particular concept (or role) to the label of a specific node (or edge), or the detection of a contradiction (clash) in the label of a node. Glass box approaches, while efficient, are reasoner and proof procedure dependent. However capable a reasoner one selects, there will be ontologies which it cannot handle, or handle as well as alternatives. Furthermore, glass box approaches require extensive revision as new features are added to the reasoner. Given these drawbacks, black box approaches are still worth exploring.

The intuition behind our black box approach is simple: given a concept $C$ unsatisfiable relative to $\mathcal{T}$, add axioms from $\mathcal{T}$ to a freshly generated ontology $\mathcal{T}'$ until $C$ is found unsatisfiable relative to $\mathcal{T}'$. We then prune extraneous axioms in $\mathcal{T}'$ until we arrive at a single minimal justification. Thus, the algorithm consists of two stages: (i) "expand" $\mathcal{T}'$ to find a superset of a justification and (ii)"shrink" to find the final justification. Each stage involves various consistency-check calls to the reasoner and the main aim of optimizations should be minimizing the number of consistency tests.

This algorithm, which we refer to as SINGLE_MUPS$_{Black-Box}$(C, $\mathcal{T}$), shown in Table 4, is composed of two main parts: in the first loop, the algorithm generates an empty KB $\mathcal{T}'$ and inserts into it axioms from $\mathcal{T}$ in each iteration, until the input concept $C$ becomes unsatisfiable w.r.t $\mathcal{T}'$. In the second loop, the algorithm removes an axiom from $\mathcal{T}'$ in each iteration and checks whether the concept $C$ turns satisfiable w.r.t. $\mathcal{T}'$, in which case the axiom is reinserted into $\mathcal{T}'$. The process continues until all axioms in $\mathcal{T}'$ have been tested.

| **Algorithm**: SINGLE_MUPS$_{Black-Box}$<br>**Input**: KB $\mathcal{T}$, Unsatisfiable concept $C$<br>**Output**: KB $\mathcal{T}'$ |
|---|
| $\mathcal{T}' \leftarrow \emptyset$<br>**while** (C is satisfiable w.r.t $\mathcal{T}'$) **do**<br>  select a set of axioms $s \subseteq \mathcal{T}/\mathcal{T}'$<br>  $\mathcal{T}' \leftarrow \mathcal{T}' \cup s$<br>**for each** axiom $k' \in \mathcal{T}'$, **do**<br>  $\mathcal{T}' \leftarrow \mathcal{T}' - \{k'\}$<br>  **if** (C is satisfiable w.r.t. $\mathcal{T}'$), **then**<br>  $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{k'\}$ |

Table 4: Single MUPS (Black Box)

A key component of an efficient "expand" stage is selecting *which* axioms to copy over from $\mathcal{T}$ into $\mathcal{T}'$. In our implementation, we run a loop that starts by inserting the concept definition axioms into $\mathcal{T}'$ and slowly expands $\mathcal{T}'$ to include axioms of structurally connected concepts, roles, and individuals (i.e., axioms which share terms in their signature). We vary the pace with which the fragment $\mathcal{T}'$ is expanded, initially considering few axioms to keep the size of $\mathcal{T}'$ bounded, and later allowing a large number of axioms

into $\mathcal{T}'$ (at each iteration of the loop) if the concept continues to remain satisfiable in $\mathcal{T}'$.

Also, we perform a fast pruning of $\mathcal{T}'$ before proceeding to the "shrink" stage. The idea here is to use a window of $n$ axioms, slide this window across the axioms in $\mathcal{T}'$, remove axioms from $\mathcal{T}'$ that lie within the window and determine if the concept is still unsatisfiable in the new $\mathcal{T}'$. If the concept turns satisfiable, we can conclude that at least one of the $n$ axioms removed from $\mathcal{T}'$ is responsible for the unsatisfiability and hence we insert the $n$ axioms back into $\mathcal{T}'$. However, if the concept still remains unsatisfiable, we can conclude that all $n$ axioms are irrelevant and we remove them from $\mathcal{T}'$.

### 7.3.2 Computing All Justifications

While the the techniques we have just described can be used to find a single justification of an unsatisfiable concept, extending them to compute all the justifications is not straightforward. In particular, extending glass-box techniques to computing all justifications amounts to saturating the completion graph generated by the DL reasoner (when testing the concept satisfiability) in order to explore *all* possible clashes. This, in effect, requires us to "turn off" many of the key optimizations in the reasoner. Since the excellent performance of current OWL reasoners critically depends on these optimization techniques, having to disable them renders this technique (currently) impractical. The optimizations (such as early clash detection or backjumping) need to be reworked (if possible) to handle the fact that finding a single clash is no longer useful (in that it stops the search).

Fortunately, given an initial justification, we can use other techniques to compute the remaining ones. A plausible one is to employ a variation of the classical Hitting Set Tree (HST) algorithm. This technique is both reasoner independent and, perhaps surprisingly, practically effective.

Consider a set $U$ and a set $S \subseteq \mathcal{P}U$ of *conflict sets*, with $\mathcal{P}$ the powerset operator. The set $T \subseteq U$ is a *hitting set* for $S$ if each $s_i \in S$ contains at least one element of $T$. If, in addition, no $T' \subset T$ is a hitting set for $S$, then $T$ is a *minimal hitting set* for $S$. The *Hitting Set Problem* with input $S, U$ is to compute all the minimal hitting sets for $S$. The problem is of interest to many kinds of *diagnosis* tasks and has found numerous applications.

Given a collection $S$ of conflict sets, Reiter's algorithm constructs a labeled tree called *Hitting Set Tree* (HST). Nodes in an HST are labeled with a set $s \in S$ and: **1)** if $H(v)$ is the set of edge labels on the path from the root to the node $v$, then $\mathcal{L}(v) \cap H(v) = \emptyset$; **2)** for each $\sigma \in \mathcal{L}(v)$, $v$ has a successor $w$ and $\mathcal{L}(\langle v, w \rangle) = \sigma$; **3)** if $\mathcal{L}(v) = \emptyset$, then $H(v)$ is a hitting set for $S$. Our approach is based on the following result, that establishes the relationship between the Hitting Set and axiom pinpointing problems:

**Theorem 7.4** *Let $C$ be unsatisfiable w.r.t $\mathcal{T}$ and let $\mathcal{T}' \subset \mathcal{T}$, with $\mathcal{T}' = \mathcal{T} - \mathcal{H}$, then:*

1. *$C$ is satisfiable w.r.t. $\mathcal{T}'$ if and only if $\mathcal{H}$ is a Hitting Set for $JUST(C, \mathcal{T})$*
2. *$\mathcal{H}$ is a minimal Hitting Set for $JUST(C, \mathcal{T})$, if and only if there is no $\mathcal{H}' \subset \mathcal{H}$ such that $C$ is satisfiable w.r.t. $\mathcal{T} - \mathcal{H}'$.*

The algorithm relies on the fact that, in order to make a concept $C$ satisfiable w.r.t. $\mathcal{T}$, one needs to remove from $\mathcal{T}$ *at least* one axiom from each of the elements of $JUST(C, \mathcal{T})$. Our aim is to use this theorem and Reiter's HS algorithm to obtain the set $JUST(C, \mathcal{T})$ out of any of its elements.

The main component of the algorithm is the recursive procedure SEARCH-HST, that effectively traverses an HST.

ALL-JUST(C, $\mathcal{T}$)
**Input:** Concept $C$ and ontology $\mathcal{T}$
**Output:** Set $S$ of ontologies
(1)**Globals:** S $\leftarrow$ HS $\leftarrow \emptyset$
(2)just $\leftarrow$ SINGLE_JUST$(C, \mathcal{T})$
(3)S $\leftarrow$ S $\cup$ {just}
(4)$\alpha \leftarrow$ select some $i \in$ just
(5)path $\leftarrow \emptyset$
(6)SEARCH-HST(C, $\mathcal{T} \setminus \{\alpha\}, \alpha$, path)
(7)**return** S

SEARCH-HST(C, $\mathcal{T}, \alpha$, path)
**Input:** $C$ and $\mathcal{T}$ as before
$\alpha$ is the axiom that was removed
*path* is a set of axioms
**Output:** *none* — modifies globals S, HS
(1)**if** path $\cup \{\alpha\} \subseteq h$ for some $h \in$ HS
(2) **return**
(3)**if** $C$ is unsatisfiable relative to $\mathcal{T}$
(4) new-just $\leftarrow$ SINGLE_JUST$(C, \mathcal{T})$
(5) S $\leftarrow$ S $\cup$ {new-just}
(6) new-path $\leftarrow$ path $\cup \{\alpha\}$
(7) **foreach** $\beta \in$ new-just
(8) SEARCH-HST$(C, \mathcal{T} \setminus \{\beta\}, \beta$, new-path)
(9)**else**
(10) HS $\leftarrow$ HS $\cup$ path

The algorithm has exponential complexity and its worst case arises when all the sets in $JUST(C, \mathcal{T})$ are mutually disjoint.

## 7.4 Error Repair

owever, while the emphasis was on pinpointing and explaining the errors in OWL ontologies, there was a lack of support for (semi-)automatically repairing or fixing them. Though in most cases, repairing errors is left to the ontology modeler's (/author's) discretion, and understanding the cause of the error certainly helps make resolving it much

easier, bug resolution can still be a non-trivial task, requiring an exploration of remedies with a cost/benefit analysis, and tool support here can be quite useful.

In this section, we propose a technique to generate repair solutions automatically based on strategies used to rank erroneous axioms and a modified Reiter's Hitting Set algorithm. In addition, we consider strategies for rewriting axioms.

Note that while we focus on repairing unsatisfiable concepts in a consistent OWL Ontology, the underlying problem involves dealing with and rectifying a set of erroneous axioms, and thus the same principles for generating repair solutions are applicable when debugging an inconsistent OWL ontology.

### 7.4.1 Strategies for Ranking Axioms

We now discuss a key piece of the repair process: selecting which erroneous axiom(s) to remove from the MUPS in order to fix the unsatisfiable concepts.

For this purpose, an interesting factor to consider is whether the axioms in the MUPS can be *ranked* in order of importance. Repair is then reduced to an optimization problem whose primary goal is to get rid all of the inconsistency errors in the ontology, while ensuring that the highest rank axioms are preserved and the lowest rank axioms removed from the ontology.

A simple criterion to rank axioms is to count the number of times it appears in the MUPS of the various unsatisfiable concepts in an ontology. This idea is similar to the notion of *arity* of the axiom as discussed in [SC03a]. If an axiom appears in $n$ different MUPS (in each set of the MUPS), removing the axiom from the ontology ensures that $n$ concepts turn satisfiable. Thus, higher the frequency, lower the rank assigned to the axiom.

Besides the axiom frequency in the MUPS, we consider the following strategies to rank ontology axioms:

- impact on ontology when the axiom is removed or altered (need to identify *minimal impact* causing changes),
- test cases specified manually by the user to rank axioms,
- provenance information about the axiom (author, source reliability, time-stamp etc.), and
- relevance to the ontology in terms of its usage

The basic notion of revising a knowledge base while preserving as much information as possible has been discussed extensively in belief revision literature. We now apply the same principle to repairing unsatisfiable concepts in an OWL ontology, i.e., we determine the impact of the changes made to the ontology in order to get rid of unsatisfiable concepts, and identify minimal-impact causing changes. Since repairing an unsatisfiable concept involves removing axioms in its MUPS, we consider the impact of axiom removal on the OWL ontology.

A fundamental property of axiom removal based on the monotonicity of OWL-DL is the following: *removing an axiom from the ontology cannot add a new entailment.*

Hence, we only need to consider entailments (subsumption, instantiation etc.) that are lost upon axiom removal, and need not consider whether other concepts in the ontology turn unsatisfiable.

For now, we shall only consider subsumption/disjointness (between atomic concepts) and instantiation (of atomic concepts) as the only interesting entailments to check for when an axiom is removed. In the next subsection, we discuss how the user can provide a set of test cases as additional interesting entailments to check for.

As mentioned earlier, our Axiom Pinpointing service computes the minimal set of axioms (justification) responsible for any arbitrary entailment of an OWL-DL ontology . Thus, we can use this service to compute the justification sets for the significant subsumption and instantiation relationships in the ontology. When removing an axiom, we can check if it falls into a particular justification set, and accordingly determine which subsumption and/or instantiation relation(s) would break directly. Axioms to be removed can then be ranked based on the number of entailments they break (higher the rank, lesser the entailments broken).

An important distinction is the entailments resulting from the unsatisfiable concepts in the ontology. Note that when a concept is unsatisfiable, it is equivalent to the bottom concept (or in OWL lingo, `owl:Nothing`), and hence is trivially equivalent to all other unsatisfiable concepts, and is a subclass of all satisfiable concepts in the ontology. In this case, we need to differentiate between the stated or explicit entailments related to unsatisfiable concepts and the trivial ones. Thus, we apply the following strategy: if a given entailment related to an unsatisfiable concept holds in a fragment of the ontology in which the concept is satisfiable, we consider the entailment to be explicit.

There are two techniques to obtain such explicit entailments: the first is a brute-force approach that involves considering all possible (minimal) solutions to fix the unsatisfiable concept in the ontology, and verifying if the entailment still holds in the modified ontology. In order to obtain minimal repair solutions, we can use Reiter's algorithm as seen in the next section. On the other hand, the second approach is much faster (though incomplete) and is based on using the *structural analysis* techniques seen in [KPSH05] to detect the explicit relationships involving unsatisfiable concepts without performing large scale ontology changes. For example, we can use the *Ontology Approximation* heuristic to get rid of the contradictions in the ontology while revealing the hidden subsumption entailments.

Having obtained the explicit entailments related to unsatisfiable concepts, we can present them to the user to learn which, if any, of the relationships are (un)desired. This information would then be used in the plan generation phase.

We consider a few examples that highlight the significance of this strategy.

In addition to the standard entailments considered in the previous subsection, the user can specify a set of test cases describing desired entailments. Axioms to be removed can be directly ranked based on the desired entailments they break.

Also, in some cases, the user can specify *undesired* entailments to aid the repair process. For example, a common modeling mistake is when an atomic concept $C$ inadvertently becomes equivalent to the top concept, `owl:Thing`. Now, any atomic concept disjoint from $C$ becomes unsatisfiable. This phenomenon occurred in the CHEM-A ontology, where the following two axioms caused concept $A$ (anonymized) to become equivalent

to `owl:Thing`: $\{A \equiv \forall R.C,\ domain(R, A)\ \}$. Here, specifying the undesired entailment prevented our ontology-effect strategy from considering the impact of removal of the erroneous axiom (in this case, the equivalence, which needed to be changed to a subclass) on this entailment.

**Provenance Information regarding Change:** Provenance information about an axiom can act as a useful pointer for determining its importance/rank, i.e., based on factors such as:

- reliability of the source (author, document etc.),
- context/reason for which the axiom was added (specified as an annotation or otherwise), and
- time the axiom was specified.

OWL has support for adding human-readable annotations to entities in an ontology using `owl:AnnotationProperties` such as *rdfs:label*, *rdfs:comment*. However, there is no direct provision to annotate assertions or axioms in the ontology, unless one resorts to reification. In general, manually providing provenance information about axioms can be a tedious task, and thus tool support is critical. To address this issue, ontology editors such as Protege [GMF+03], KAON [MS06] and Swoop have the option to maintain an elaborate change log to record provenance information.

In Swoop, we automatically keep track of all changes made to an OWL ontology, storing information such as authorship, date etc of each change. Additionally, we use a *change-ontology* that represents various atomic and complex change operations to serialize the change-log to RDF/XML, which can then be shared among users.

Such information is extremely useful for ranking axioms in a collaborative ontology building context, i.e., if a group of authors are collectively building an ontology, and there exists a precedence level among the authors, i.e., ontology changes made by the supervisor are given higher priority than those made by a subordinate. In this case, for each change made, one can derive the corresponding axioms added to the ontology, and automatically determine the rank of each axiom based on the person making the change.

**Syntactic Relevance:** There has been research done in the area of ontology ranking [DPF+05], where for example, terms in ontologies are ranked based on their structural connectedness in the graph model of the ontology, or their popularity in other ontologies, and the total rank for the ontology is assigned in terms of the individual entity ranks. Since an ontology is a collection of axioms, we can, in theory, explore similar techniques to rank individual axioms. The main difference, of course, lies in the fact that ontologies as a whole can be seen as documents which link to (or import) other ontology documents, whereas the notion of linkage is less strong for individual axioms.

Here, we present a simple strategy that ranks an axiom based on the *usage* of elements in its signature, i.e., for each OWL entity (atomic class, property or individual) in the signature of the axiom, we determine how often the entity has been referenced in other axioms in the ontology, and sum the reference counts for all the entities in the axiom

signature to obtain a measure of its syntactic (or structural) relevance. The significance of this strategy is based on the following intuition: if the entities in the axiom are used (or are referred to) often in the remaining axioms or assertions of the ontology, then the entities are in some sense, core or central to the overall theme of the ontology, and hence changing or removing axioms related to these entities may be undesired. For example, if a certain concept is heavily instantiated, or if a certain property is heavily used in the instance data, then altering the axiom definitions of that concept or property is a change that the user needs to be aware of. Similarly, in large ontologies where certain entities are accidentally underspecified or unused, axioms related to these entities may be given less importance.

The simple strategy presented above can be altered in various ways such as by restricting usage counts to certain axiom types, and/or weighing certain kinds of axioms differently than others (e.g., weighing property attribute assertions such as `InverseFunctional` higher). This would be motivated by user preferences depending on the ontology modeling philosophy and purpose.

### 7.4.2 Generating Repair Solutions

So far, we have devised a procedure to find tagged MUPS for an unsatisfiable concept in an OWL-DL ontology and proposed various strategies to rank axioms in the MUPS. The next step is to generate a repair plan (i.e., a set of ontology changes) to resolve the errors in a given set of unsatisfiable concepts, taking into account their respective MUPS and axiom ranks.

**Modifying Reiter's Algorithm:** The first technique we propose for generating repair plans is to modify the Reiter's Hitting Set algorithm that we have previously introduced in this section to take into account the axiom ranks.

However, there is a drawback of using the above procedure to generate repair plans, i.e., impact analysis is only done at a single axiom level, whereas the cumulative impact of the axioms in the repair solution is not considered. This can lead to non-optimal solutions.

In order to resolve this issue, we propose another modification to the algorithm above: each time a hitting-set $HS$ is found, we compute a new path-rank for $HS$ based on the cumulative impact of the axioms in the hitting-set. The algorithm now finds repair plans that minimize these new path-ranks. Note that the early termination condition for paths remains the same since the path rank represents a lower bound, as cumulative impact is always greater than or equal to the sum of individual unique impacts.

**Improving and Customizing Repair:** The hitting set algorithm can be used in general to fix any arbitrary set of unsatisfiable concepts, once the MUPS of the concepts and the ranks for axioms in the MUPS is known. Thus, a brute force solution for resolving *all* the errors in an ontology involves determining the MUPS (and ranking axioms in the MUPS) for *each* of the unsatisfiable concepts. This is computationally expensive and moreover, unnecessary, given that strong dependencies between unsatisfiable concepts may exist. Thus, we need to focus on the MUPS of the critical or root contradictions in the ontology.

To achieve this, we make use of a debugging service we have devised in [KPSH05] that identifies the *root* unsatisfiable concepts in an ontology, which propagate and cause errors elsewhere in the ontology, leading to *derived* unsatisfiable concepts. Intuitively, a root unsatisfiable concept is one in which a clash or contradiction found in the concept definition does not *depend on the unsatisfiability* of another concept in the ontology; whereas, a derived unsatisfiable concept acquires a contradiction due to its dependence on another unsatisfiable concept. For example, if $A$ is an unsatisfiable concept, then a concept $B$ ($B \sqsubseteq A$) or $C$ ($C \sqsubseteq \exists R.A$) also becomes unsatisfiable due to its dependence on $A$, and is thus considered as derived.

We have experimented with the root/derived debugging service on numerous OWL ontologies that have a large number of unsatisfiable concepts and found it to be useful in narrowing down the error space quickly, e.g, for the Tambis OWL Ontology, only 3 out of 144 unsatisfiable concepts were discovered as roots in under 5 seconds. From a repair point of view, the key advantage here is that one needs to focus on the MUPS of the root unsatisfiable concepts alone since fixing the roots effectively fixes a large set of directly derived concept bugs.

Also, the service guides the repair process which can be carried out by the user at three different granularity levels:

- *Level 1: Reparing a single unsatisfiable concept at a time*: In this case, it makes sense to deal with the root unsatisfiable concepts first, before resolving errors in any of the derived concepts. This technique allows the user to monitor the entire debugging process closely, exploring different repair alternatives for each concept before fully fixing the ontology. However, since at every step in the repair process, the user is working in a localized context (looking at a single concept only), the debugging of the entire ontology could be prolonged due to new bugs introduced later based on changes made earlier. Thus, the repair process may not be optimal.

- *Level 2: Repairing all root unsatisfiable concepts together*: The user could batch repair all the root unsatisfiable concepts in a single debugging iteration before proceeding to uncover a new set of root/derived unsatisfiable concepts. This technique provides a cross between the tool-automation (done in level 3) and finer manual inspection (allowed in level 1) with respect to bug correction.

- *Level 3: Repairing all unsatisfiable concepts*: The user could directly focus on removing all the unsatisfiable concepts in the ontology in one go. This technique imposes an overhead on the debugging tool which needs to present a plan that accounts for the removal of all the bugs in an optimal manner. The strategy works in a global context, considering bugs and bug-dependencies in the ontology as a whole, and thus may take time for the tool to compute, especially if there are a large number of unsatisfiable concepts in the ontology (e.g. Tambis). However, the repair process is likely to be more efficient compared to level 1 repair.

The number of steps in the repair process depends on the granularity level chosen by the user: for example, using Level 1 above, the no. of steps is atleast the no. of unsatisfiable concepts the user begins with; whereas using Level 3 granularity, the repair

reduces to a single big step. To make the process more flexible, the user should be allowed to change the granularity level, as and when desired, during a particular repair session.

### 7.4.3 Suggesting Axiom Rewrites

Now, to make our repair solution more flexible, we consider strategies to rewrite erroneous axioms instead of strictly removing them from the ontology. Note that rewriting an axiom involves an axiom removal followed by an addition. Thus, similar to the impact analysis performed for axiom removal, we also need to consider entailments that are introduced when an axiom is added. Currently, we only check if unsatisfiable concepts arise upon axiom addition.

**Using Erroneous Axiom Parts:** As shown in section 3.2, our Axiom Pinpointing service has been extended to identify parts of axioms in the MUPS responsible for making a concept unsatisfiable. Having determined the erroneous part(s) of axioms, we can suggest a suitable rewrite of the axiom that preserves as much as information as possible while eliminating unsatisfiability.

**Identifying Common Pitfalls:** Common pitfalls in OWL ontology modeling have been enumerated in literature [ALRR04]. We have summarized some commonly occurring errors that we have observed (in addition to those mentioned in [ALRR04]), highlighting the *meant* axiom and the reason for the mistake in each case.

| **Asserted** | **Meant** | **Reason for Misunderstanding** |
|---|---|---|
| $A \equiv C$ | $A \sqsubseteq C$ | Difference between Defined and Primitive concepts |
| $A \sqsubseteq C$<br>$A \sqsubseteq D$ | $A \sqsubseteq C \sqcup D$ | Multiple subclass has intersection semantics |
| `domain(P,A)`<br>`range(P,B)` | $A \sqsubseteq \forall P.B$ | Global vs. Local property restrictions |
| `domain(P,A)`<br>`domain(P,B)` | domain(P, $A \sqcup B$) | Unclear about multiple domain semantics |

A library of error patterns can be easily maintained, extended and shared between ontology authors using appropriate tool support. Once we have identified the axioms in the ontology responsible for an unsatisfiable concept, we can check if any of the axioms has a pattern corresponding to one in the library, and if so, suggest the *meant* axiom to the user as a replacement. We note that in a lot of cases that we have observed, the most common reason for unsatisfiability is the accidental use of equivalence instead of subsumption.

In some cases, an additional heuristic to consider is the label (or ID) of the concept or role, which acts as a pointer to its intended meaning and can be used to detect mismatches in modeling. For example, the unsatisfiable concept `OceanCrustLayer` seen earlier in the Sweet-JPL OWL ontology was accidentally defined to be a subclass of `CrustRegion`, instead of `CrustLayer`.

A combination of the heuristics was used to debug an error in the University ontology. The concept `ProfessorInHCIorAI` was responsible for the unsatisfiable concepts `AI_Student` and `HCI_Student` because there were two separate subclass axioms for `ProfessorInHCIorAI`, associating it with the student concepts separately, whereas the 'or' in the concept name implied that a disjunction was intended.

## 7.5 Glass-box algorithms

Glass-box algorithms are built modifying the existing tableau-based decision procedures built in a reasoner. The adaptations needed for identifying a set of axioms relevant to a given error in an ontology, have been seen as strongly dependent on the DL under consideration and the current implementation used. Thus, Glass-box algorithms are typically custom-made. The idea undelying all these adjustments is, nonetheless, always the same.

Tableau algorithms consist on a repeated application of rules over sets of assertions, or ABoxes. A rule verifies whether an ABox contains a particular subset of assertions, and whether some axioms exist in the input TBox. If they do, then the rule *triggers*. It then makes possibly multiple copies of the selected ABox and adds new assertions to each of these copies. During the execution of this algorithm, no assertion is ever removed from an ABox.

In order to find the relevant causes of an inconsistency, the modified algorithm must keep track of the axioms ultimately responsible for the addition of an assertion to the ABox. To achieve this, a Boolean monotonic formula can be associated to each assertion. First, each axiom is represented by a distinct propositional variable and each assertion in the input ABoxes is associated to a tautology. When a rule is triggered, every new assertion added to the set will be linked to the conjunction of the axioms and the formulas of assertions that triggered the rule. Furthermore, if an assertion can be produced in different ways, it will be tied to the formula consisting of the disjunction of the formulas that produce it.

The formulas attached to the assertions have the following property. If an assertion $a$ is linked to the formula $\phi$, then, for every valuation $\omega$, $\omega$ maps $\phi$ to true if and only if it holds that if the tableau is run using only the axioms that are also evaluated to true by $\omega$, then the assertion $a$ will also be produced. In other words, the formulas do represent the axiomatic cause for each assertion to be added to an ABox. These formulas are then used to produced a so-called *clash-formula*: a Boolean monotonic formula representing the axiomatic causes of inconsistency.

The clash-formula provides all the information needed to find both, minimal justifications for the inconsistency, and maximal sub-TBoxes solving the inconsistency found. For the former one needs only to find minimal valuations that make the clash-formula true, while for the latter, the task consists in finding maximal valuations mapping the formula to false. Since the clash-formula is monotonic, the task of finding such minimal and maximal valuations does make sense.

We have shown this simply idea is applicable to every tableau-based algorithm without blocking conditions. To implement it in a reasoner, it is only necessary to extend it to associate a label to every assertion, and compute some simple operations over these labels. In other words, we have been able to describe a simple method for modifying an

implemented tableau-based algorithm to allow for pinpointing the causes of inconsistencies, if these are found. Although this method is a Glass-box approach, since it requires a modification in the current implemented algorithm, it is a general approach that can be applied to any current, or future, tableau-based reasoning service.

The importance of this Glass-box approach relies on the fact that it makes no assumptions on the tableau used; it simply states a method to modify the original tableau algorithm in order to obtain also the causes of inconsistency if it is found during the application of the algorithm. Hence, this approach can be used regardless of the underlying DL or the reasoning task for which the tableau algorithm is used. In other words, whenever a tableau algorithm is available for deciding the presence of an error, then the Glass-box approach can be used for pinpointing the main causes and correcting such found error, regardless of the definition of error used, and of the underlying logic.

Apart from soundness and completeness, two of the main characteristics desirable in every implemented tableau-based algorithm are:

- *Termination.* The algorithm will give an answer after finite time, for every possible input.
- *Application order irrelevance.* The order in which the rules are applied to the ABoxes has no effect on the final result.

Termination is important to ensure that an answer will be given to the user, regardless of the input it states. The irrelevance of the application order opens the doors for finding orderings that require minimal number of steps to terminate, optimizing this way the execution time. It can also be used to ensure termination since, while for every tableau-based algorithm the order in which the rules are applied is irrelevant for the answer, some of these algorithms might not be terminating.

The properties of the clash-formula ensure that the modification of the original tableau method will preserve soundness and completeness, and hence, will always give the correct answer, if the original algorithm did so too. Even with this result, the modified algorithm would be useless in practice if any of the two properties, termination and application order irrelevance, was not preserved. If termination is lost, then an answer might never be returned for a given input; if the application order becomes relevant for finding the correct answer, then not only the order-dependent optimizations turn to be useless, but also every application order must be checked, inducing an exponential-step blow-up in the execution time.

Under the minimal assumption that the input TBox and the initial ABoxes are finite, it is easy to show that the algorithm obtained by the modifications we described earlier preserves termination. This result follows from the fact that, if the original algorithm terminates, then it will produce only finitely many finite ABoxes. Since every rule application adds at least one element to an ABox, then all the elements will be produced after finitely many steps. The modified algorithm can then perform extra-steps concerning the different ways that each element can be generated, but each of these extra steps always associates a more general formula to the assertions. Since there are only finitely many propositionally-distinct Boolean monotonic formulas, these extra steps can be done only finitely, and hence, the new algorithm also terminates.

The irrelevance of the application order is also preserved by this modified algorithm. This is a simple consequence from the fact that, whenever an assertion is added to an ABox, it is never eliminated again. Thus, whenever a rule is applicable, it will still be applicable in any point in the future until it is applied. Thus, when there are no more applicable rules, and the algorithm stops, the elements that would be added by that rule application must be already present in the ABox, or, in other words, the rule was already applied.

For expressive DLs, or TBoxes containing GCIs, it is sometimes necessary to add *blocking conditions* to some rules. These are conditions that stop the application of a rule even when it is triggered. Blocking conditions vary greatly depending on the properties in the DL decided by the algorithm. Due mainly to this variety, the notion of blocking condition has not yet been formalized and the preservation of the forementioned properties is yet to be shown for the algorithm obtained by modifying in the same fashion these kinds of tableaus.

In order for the modified algorithm to work properly, there is a condition that must be satisfied by the original tableau-based procedure. This condition, called axiom monotonicity, states that the addition of axioms can never remove an inconsistency. When no blocking conditions are present, the tableau is always axiom monotonic. Depending on what the definition of blocking condition is, it might be the case that a tableau using *valid* blocking conditions is not axiom monotonic. In that case, it cannot be ensured that the modified algorithm works in a correct manner. For this reason, the minimal semantic property required in a tableau containing blocking conditions is that it is axiom monotonic. Although this property is enough for ensuring correctness of the modified method, semantic properties are usually not desired, since they cannot be easily verified for a given tableau-based algorithm.

One thing to be noticed is that blocking conditions are generally used, in DL reasoning, only to ensure termination. With this in mind, it should be possible to define adequate conditions that involve the goal of termination and still ensure axiom monotonicity, while being easily checked. Such conditions would extend the pinpointing methodology to tableaus that require such blocking conditions to ensure termination, and should, of course, be required that the modified algorithm preserves such termination property. Current research is focused on finding easily-verifiable conditions on the tableaus with blocking conditions which ensure that extending them to use the described labeling-based Glass-box approach will lead to a correct pinpointing algorithm.

## 7.6 Related Work

The Ontology Engineering community widely agrees on the importance of explaining the output of a reasoner to the user in a sensible way. The first steps in this direction, in the context of Description Logics, were taken in the early and mid nineties by the developers of the CLASSIC system. The explanation component in CLASSIC [McG96] generated formal proofs for an inference using a deductive framework based on 'natural semantics' style proof rules that needed to be explicitly stated for the DL. CLASSIC incorporated a structural subsumption algorithm and was able to generate explanations for concept subsumption [McG96]. In the late nineties, the authors of [BFH+99] used a modified

sequent calculus to explain tableau-based subsumption proofs for the logic $\mathcal{ALC}$.

A different view was recently explored in [SC03a], where the trace of a tableau reasoner was exploited to maintain a dependency relation between axioms in the KB and the inferences drawn from it. The motivation was debugging unsatisfiable concepts in the DICE terminology and the work applied to unfoldable $\mathcal{ALC}$ TBoxes. The paper provided a formalization of the problem based on the notion of *Minimal Unsatisfiability Preserving Sub-TBoxes* (MUPS). Roughly, a MUPS for an atomic concept A is a minimal fragment of the KB in which A is unsatisfiable. Obviously, a concept may have several different MUPS within an ontology.

In [KPSH05], we extended this technique to the more expressive Description Logic $\mathcal{SHIF}$, provided an optimized implementation in the reasoner Pellet and a UI in the ontology editor Swoop. We showed through a user study that these techniques are effective for debugging unsatisfiable concepts and proposed various enhancements in the UI to improve the explanation.

# 8 Model Checking and Model Generation

## 8.1 Introduction

Model checking as well as model generation (aka model finding) are well-established methodologies for formally verifying properties of possibly time-evolving systems. A recent survey can be found in [Hut06]. Usually, some aspects of real-world systems have to be abstracted away in order to make them accessible to formal logical modeling: continuous vs. discrete behavior, granularity, stochasticity, etc. Nevertheless, model-checking tools are successfully applied in practice. Indeed, improvements in the underlying decision procedures (most notably SAT and BDDs) together with higher-level specification languages have broadened the applicability of these techniques. New application fields have been identified recently. One such field comprises solving selected problems arising in ontology management and evolution as a complement to dedicated DL engines.

The *model generation problem* is postulated as follows. Given an ontology $O$, which is a pair $(\mathcal{T}, \mathcal{A})$ of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, find an interpretation $\mathcal{I}$ which satisfies all axioms of $\mathcal{T}$ and $\mathcal{A}$. In case of a *model checking problem* the goal is to prove whether a given interpretation is a model.

In order to support the ontology development process in an incremental way, our thesis is that well-known model-generation tools can be adopted accordingly and provide major benefits for human ontology designers. In this chapter we evaluate pros and cons of applying an existing model checking and generation tool in this context (see below for details).

In fact, the ontology designer is often not interested in just testing the satisfiability of an ontology by checking whether one single model exists, but possibly wants to inspect a number of generated models instead. This way, unintended models might be identified. This kind of modeling methodology has been proven to be very effective in software engineering (e.g., [Jac06]). The ontology designer should be offered a possibility to adjust the ontology by examining automatically generated relational model structures. Model generators support this process quite well whereas for checking the satisfiability of ontologies and computing the taxonomy tableau algorithms have been proven to be very effective. Thus, it seems attractive to augment tableau provers to also support model generation. Current tableau algorithms are not well applicable as model generation procedures since they only return (a description of) a so-called single canonical model [BCM$^+$03a]. Instead, model finders are able to enumerate all models systematically. This can indeed be useful for ontology design tasks. To illustrate this, we discuss the following simple example. Let $A, B$ be concepts and $R$ be a role. Suppose the satisfiability of the following concept is checked by both a DL system and a model finder:

$$(\exists R.A) \sqcap (\exists R.B)$$

The model generated by a tableau algorithm is shown on the left-hand side of the vertical bar in Figure 8. However, the ontology designer may be more interested in inspecting models computed by a model finder (see Figure 8, to the right of the vertical bar). The latter four models are not considered by the rules of tableau prover because if the left structure is model, then the structures to the right of the bar are also models.

Thus, it suffices to consider only one model (the one to the left of the bar) in order to show satisfiability. In order to evaluate an ontology (i.e., the concepts, roles, and axioms in it), considering all models is nevertheless interesting as we argued above. Thus, it makes sense to investigate contemporary model finders, study the state of the art in this field from a practical point of view, and identify possible limitations.

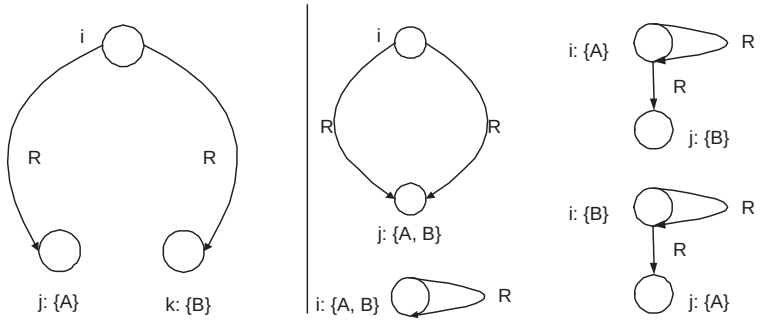


Figure 8: Models of $(\exists R.A) \sqcap (\exists R.B)$

In our case study, we adopt a particular finite model finder, namely Alloy Analyzer 3.0 [Jac06], whose language is based on relational calculus and thus allows for straightforward representation of $\mathcal{ALC}$ knowledge bases. The technique reported here is not the first attempt at rephrasing ontology languages in Alloy, yet it has been developed independently. In [WDSS06], case studies have been published where an ontology has been formulated (and further analyzed) in Alloy, Z, and (recently) HOL. Unlike our approach to capture the semantics of $\mathcal{ALC}$ constructs directly, the authors define a translation schema that considers the meta-level of the ontology language in terms of individuals for concepts and properties as well as relationships among these individuals.

In the context of the common ontology framework proposed by TONES, we concern DL-based stand-alone ontologies in the design phase. As argued by [WDSS06], a synergy between DL systems and model-generating tools results in a new reasoning service, a model finding service, which takes a knowledge base as input and presents generated models as output.
We show results achieved so far for several case studies:

- Model inspection (Sect. 8.3.1);
- Visual display of counterexamples for a subsumption assumption (Sect. 8.3.2) and for a concept equivalence assumption (Sect. 8.3.3);
- Finding models for $\mathcal{ALC}$ terms whose satisfiability analysis is expensive for contemporary tableaux-based reasoners (Sect. 8.4).

In the next section we address the translation rules for DL into Alloy, starting with the base logic $\mathcal{ALC}$.

## 8.2 Translation from Description Logics into Alloy

The logic underlying the Alloy Analyzer is Relational Logic (RL) whose syntax, type rules and semantics are described in [Jac00]. This logic is more than a syntactical variation of first-order logic, because it includes transitive closure. An automatic model finder requires the specification of a *scope*, a bound on the number of atoms in the universe (cardinalities of concepts). This limitation is not as dramatic as it might seem, given the so-called *small-scope hypothesis*:

> "First-order logic is undecidable, so our analysis cannot be a decision procedure: if no model is found, the formula may still have a model in a larger scope. Nevertheless, the analysis is useful, since many formulas that have models have small ones." [Jac00]

Moreover,

> "Given a relational formula, we can construct a boolean formula that has a model exactly when the original formula has a model in some given scope." [Jac00]

Given that $\mathcal{ALC}$ exhibits the finite model property, it is thus amenable to circumvent the finite-scope limitation. In fact, we can compute worst case concept cardinalities according to the maximum concept branching factor and the maximum depth of nested existential quantifiers.

### 8.2.1 Translation Rules for $\mathcal{ALC}$

**Definition 1 (Alloy Translation Rules for $\mathcal{ALC}$ Concepts)** If $A$ is a concept name, $C$, $D$ are concepts, $R$ is a role name, the following translation rules can be applied to $\mathcal{ALC}$ concepts in order to obtain semantically equivalent Alloy formulas:

| | |
|---|---|
| $A$ | `A` |
| $C \sqcap D$ | `C & D` |
| $C \sqcup D$ | `C + D` |
| $\neg C$ | `univ - C` |
| $\forall R.C$ | `univ - (R.(univ - C))` |
| $\exists R.C$ | `R.C` |

Here, `A`, `C`, `D`, `R` denote Alloy relations, `&`, `+`, `-` are set operators (intersection, union and difference, respectively), "`.`" stands for the relational join operator. The unary relation `univ` represents the set containing every instance of the universe (interpretation domain).

**Definition 2 (Alloy Translation Rules for $\mathcal{ALC}$ TBox and ABox axioms)** We summarize translation rules for $\mathcal{ALC}$ terminological and assertional axioms into Alloy.

- In $\mathcal{ALC}$, expressions $\top$ (universal concept) and $\bot$ (unsatisfiable concept) are used as abbreviations for $A \sqcup \neg A$ resp. $A \sqcap \neg A$, where $A$ is a concept name. In Alloy, we define the `TOP` relation as subset of `univ` and `BOTTOM` as subset of `TOP` containing no instances:

```
abstract sig TOP
sig BOTTOM in TOP {} fact { #BOTTOM = 0 }
```

A signature (denoted as `sig`) introduces a set of atoms. The declaration `sig A { }` introduces a set named `A`. An `abstract` signature has no elements except those belonging to the extension of its subsignatures.

- Elementary descriptions are atomic concepts and atomic roles.

  Atomic concepts are declared in Alloy as non-empty subsets of `TOP`. For example, $A$ is declared as atomic concept:

  ```
  sig A in TOP {} fact { }
  ```

  Atomic roles are specified in Alloy with a set `TOP` as both domain and range. For example, the role `hasChild` is an atomic role:

  ```
  abstract sig TOP { hasChild : set TOP }
  ```

- If $C$ is an atomic concept and $D$ is a concept, then $C \sqsubseteq D$ is called generalized concept inclusion, or GCI. GCIs are translated into Alloy using the set operator `in` (subset):

  ```
  fact {C in D}
  ```

- Concept definitions of the form $A \equiv C$, where $A$ is an atomic concept, are called equality axioms. Equalities are translated using Alloy's set equality operator `=`:

  ```
  fact { A = C }
  ```

- Instances of a given concept description are called individuals. If $i$ is an individual, then it can be defined in Alloy as follows:

  ```
  sig i in TOP {} fact { #i = 1 }}
  ```

  In Alloy, a multiplicity keyword placed before a signature declaration constrains the number of elements in the signature's set. E.g., the keyword `one` allows for defining a signature whose set contains exactly one element. Thus,

  ```
  one sig i in TOP {}
  ```

  declares instance i, having the same effect as the specification above.

  To implement the unique name assumption, additional constraints are generated to ensure that these singleton sets are pairwise different, for example

```
/* pairwise disjointness of individuals */
fact { no ( polyneikes & iokaste ) }
fact { no ( polyneikes & thersandros ) }
fact { no ( polyneikes & oedipus ) }
```

- If $a, b$ are individual names, $C$ is a concept and $R$ is a role name, than the following assertions about named individuals can be built by using constructs above:

  $C(a)$ (concept membership assertion)

  $R(a, b)$ (role membership assertions)

  In terms of Alloy we use the following transformation schema for concept membership assertions and role membership assertions resp. (`->` denotes the relational product operator):

```
fact { a in C }

fact { a -> b in R }
```

### 8.2.2 Translation of $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$

Alloy's underlying logic is expressive enough to encode $\mathcal{SHIQ}$ or even $\mathcal{SROIQ}$ formulas. As an outlook, tables below depict the Alloy formulation of $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$ concepts and role constructors as well as of additional role constructs possible in Alloy. Here, `:>` denotes the range restriction and `~` is the relational transpose operator defined over binary relations. The operator `#` applied to a relation gives the cardinality of the relation as an integer value. The binary relation `iden` relates all the instances of the universe to themselves.

| $\mathcal{SHIQ}$ concepts | Alloy translation |
|---|---|
| $\leq nR.C$ | `{ a : univ \| #(a.(R :> C)) =< n }` |
| $\geq nR.C$ | `{ a : univ \| #(a.(R :> C)) => n }` |
| inverse | `~R` |

| $\mathcal{SROIQ}$ concepts | Alloy translation |
|---|---|
| $\{o\}$ | `sig i in TOP { } fact { #o = 1 }}` |
| $\exists R.Self$ | `(R & iden).univ` |

| Further role terms | Alloy translation |
|---|---|
| $R \sqcap S$ | `R & S` |
| reflexive transitive closure | `*R` |

$\mathcal{SHIQ}$ allows for defining role hierarchies, which is a finite set of role inclusion axioms $R \sqsubseteq S$ where $R$ and $S$ are roles, and transitive roles ($R \circ R \sqsubseteq R$). In Alloy, we achieve the same expressibility using the set operator `in` and the relational composition (join) operator `.`:

```
R in S

(R . R) in R
```

In $\mathcal{SROIQ}$, a role inclusion axiom is of the form $w \sqsubseteq R$, where $w$ is a finite string of roles (e.g., $S1$, $S2$) and $R$ is a role name. The appropriate translation into Alloy is:

```
(S1 . S2) in R
```

### 8.2.3 Translation Algorithm

As input language a subset of the KRSS format is adopted. A grammar for it was specified in terms of the JavaCC parser generator [Kod04]. The generated parser contains actions to instantiate AST (Abstract Syntax Tree) nodes from Java classes, these Java classes were in turn also generated following the recommended methodology to provide Eclipse-based tooling for custom DSLs (Domain Specific Languages). The AST classes are specified in a so-called language metamodel (see Figure 9), in our case this metamodel is expressed in EMOF (Essential MOF [Gro06]) which can be considered as a subset of the language used to express UML class models. A possibility exists to enrich the metamodel with validation checks beyond syntactic correctness (so called static semantics) in the form of OCL (Object Constraint Language) invariants. These OCL expressions can be automatically translated into Java, further cutting down on the tooling development effort.

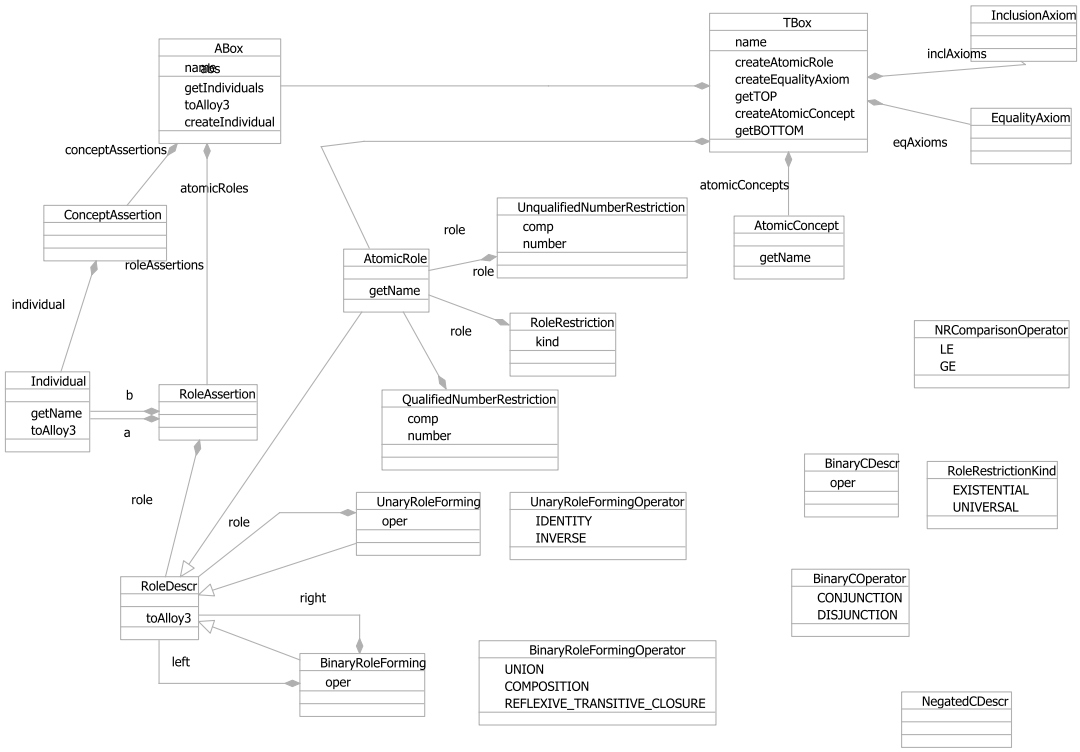


Figure 9: AST classes for object-oriented ASTs of $\mathcal{ALC}$

Another advantage of the metamodel approach is the loose coupling between concrete and abstract syntax. In effect, handling alternative input formats (e.g., those originating in W3C standards) requires generating a new Java-based parser yet the rest of the translation is isolated from this change, as only data structures specified in the metamodel are

required. In turn, extending the translation to handle, for example, $\mathcal{SHIQ}$ can leverage object-oriented extensions mechanisms to define a metamodel taking that for $\mathcal{ALC}$ as starting point. In short, most of the Java code infrastructure need not be re-developed from scratch when migrating to another DL logic.

After the ASTs of an $\mathcal{ALC}$ TBox and a (possibly empty) ABox have been built, the translation process begins with an Alloy file (`.als` extension) generated after visiting the AST nodes. Every output is made to contain Alloy definitions for $\top$ and $\bot$, followed by those for atomic concepts and atomic roles. The object-oriented structure of the AST simplifies processing at this point. The translator leverages the type system of Java 5 (in particular, parametric polymorphisms) resulting in improved readability and increased confidence in the reliability of the implementation. For example, a user of our framework can find methods such as `getIndividuals()` in class ABox, which returns a value of type `Set(Individual)`. Java 5 combines the advantages of a statically and strongly typed language with expressive types.

The assumptions that Alloy can make when searching for models are encoded as `facts`. For example, equality axioms are generated by our translator as follows:

```
res.append(NL + "/* equality axioms */" + NL);
for (EqualityAxiom ea : getTbox().getEqAxioms()) {
    res.append("fact { (" + ea.getLeft().toAlloy3() + ") = ("
        + ea.getRight().toAlloy3() + ") } ");
    res.append(NL);
}
```

The above code has been simplified to unclutter some optimizations, e.g., string manipulation is performed instead of directly writing to a buffered stream.

Similar steps are followed for inclusion axioms, concept membership assertions, and role membership assertions.

## 8.3 Case Studies

In follows, we illustrate advantages of our proposal in the context of ontology design by discussing several case studies.

### 8.3.1 Model Inspection by Counterexample Extraction

As an introductory example of model inspection, we use the Oedipus example (see [BCM+03a, p. 73]). In this example, the following ABox with some facts about the Oedipus story is supposed:

$$\begin{array}{rr} \mathit{hasChild}(\mathit{iokaste}, \mathit{oedipus}) & \mathit{hasChild}(\mathit{iokaste}, \mathit{polyneikes}) \\ \mathit{hasChild}(\mathit{oedipus}, \mathit{polyneikes}) & \mathit{hasChild}(\mathit{polyneikes}, \mathit{thersandros}) \\ \mathit{Patricide}(\mathit{oedipus}) & \neg\mathit{Patricide}(\mathit{thersandros}) \end{array}$$

Now, we want to find out whether some individual exists that have a child that is a patricide and that itself has a child that is not a patricide. This can be seen as a problem of checking the satisfiability of the concept *hasPatricideChildWithNonPatricideChild*

declared as follows:

$$hasPatricideChildWithNonPatricideChild \equiv$$
$$(\exists hasChild.(Patricide \sqcap \exists hasChild.\neg Patricide))$$

Applying $\mathcal{ALC}$ translation rules to the Oedipus knowledge base, we obtain the following Alloy specification:

```
module oedipus

abstract sig TOP {
/* atomic roles */
    hasChild : set TOP
}
sig BOTTOM in TOP {}
fact { #BOTTOM = 0 }

/* atomic concepts */
sig hasPatricideChildWithNonPatricideChild in TOP {}
sig Patricide in TOP {}

/* individuals */
one sig polyneikes in TOP {}
one sig iokaste in TOP {}
one sig thersandros in TOP {}
one sig oedipus in TOP {}
/* pairwise disjointness of individuals */
fact { no ( polyneikes & iokaste ) }
fact { no ( polyneikes & thersandros ) }
fact { no ( polyneikes & oedipus ) }
fact { no ( iokaste & thersandros ) }
fact { no ( iokaste & oedipus ) }
fact { no ( thersandros & oedipus ) }

/* equality axioms */
fact { (hasPatricideChildWithNonPatricideChild) =
     ( (hasChild).((Patricide)&((hasChild).((univ - Patricide))))) }

/* no inclusion axioms were declared */

/* concept assertions */
fact { oedipus in Patricide }
fact { thersandros in  ( univ - Patricide )  }

/* role assertions */
fact { oedipus -> polyneikes in hasChild }
fact { iokaste -> polyneikes in hasChild }
fact { polyneikes -> thersandros in hasChild }
fact { iokaste -> oedipus in hasChild }

pred show() { #univ = 4
/* to make sure that no additional persons get into the myth */}
run show for 4
```

Alloy presents the following model (Figure 10). We will discuss next how it relates to the ontology given above. In summary, Iokaste is shown to have a patricide child (Oedipus in this model) who in turn has a non patricide child (a choice of individuals in this model).

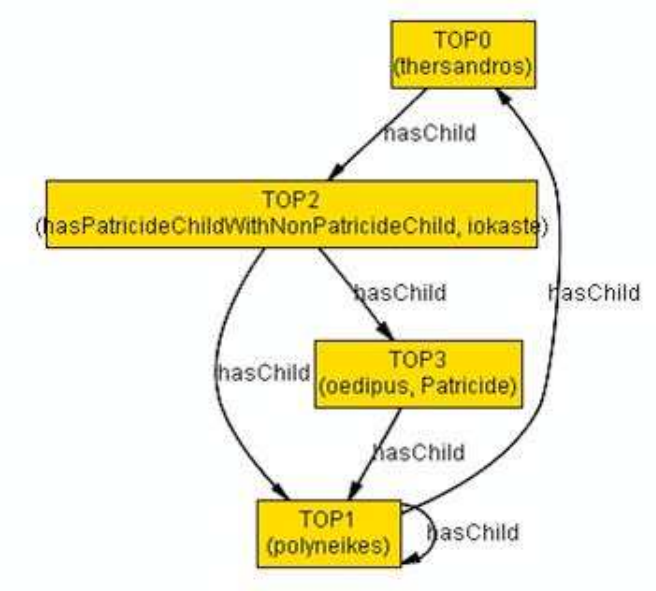


Figure 10: Oedipus example

Alloy offers a choice of customization capabilities for visualizing models but we will stick with the default settings. Particular models can be shown graphically as "snapshots" where individuals are represented as rectangles. Each such rectangle is identified by the internal name used by Alloy (e.g., "`TOP102`") which appears on the upper part of the rectangle. Arcs between rectangles stand for binary relations (roles), a label on the arc makes clear which role is being referred to. For each individual, the sets (concepts) it belongs to are shown as a comma separated list of labels on the lower part of the rectangle in question. Absence of one such labels means that the individual does not satisfy that concept. For example, the labels of the node "`TOP2`" "`iokaste, hasPatricideChildWithNonPatricideChild`" reveal that the individual `iokaste` is described by the concept `hasPatricideChildWithNonPatricideChild`. Browsing further models will show other constellations under which this concept is satisfied. Note however that in this particular model, `polyneikes` is considered to have himself as child (nothing in the TBox prevents this). Inspecting models may give rise for adjusting the ontology by adding further axioms.

### 8.3.2 Counterexamples for a Subsumption Assumption

We use the following example from [BCM+03a, p. 82] to demonstrate how a subsumption relation can be explained using Alloy. Assume that we want to check whether $(\exists r.a) \sqcap (\exists r.b)$ is subsumed by $\exists r.(a \sqcap b)$. This is equivalent to the satisfiability test of the concept *ifUnsatisfiableThenSubsumes* defined below:

$$ifUnsatisfiableThenSubsumes \equiv (\exists r.a) \sqcap (\exists r.b) \sqcap \neg(\exists.(a \sqcap b))$$

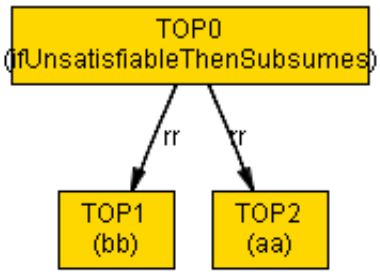


Figure 11: Counterexample for *ifUnsatisfiableThenSubsumes*

Letting Alloy analyze the predicate *ifUnsatisfiableThenSubsumes* results in several models. If left to its own devices, Alloy presents a model that minimizes the number of individuals. Alloy can be instructed however to look for models of a certain shape. We will do just that in order to display the solution presented in [BCM+03a, p. 82], which is computed by a tableau algorithm. In order to achieve this, we will constraint those model we are interested in to those having exactly three individuals, with no individual in $a$ nor $b$ having a role filler over $r$. Using Alloy syntax, we formulate the last requirement in terms of set cardinality (`#`) and the join operator, where `a.r` stands for the images over $r$ whose domain is $a$. The model we are looking for is depicted in Figure 11. The technique described above is, of course, generally applicable and results in shorter response times as only a subset of all possible models is explored.

To gain a better understanding for this result, one must recall that labels of particular individuals (nodes) contain concept names the individual belongs to. Absence of some concept name $C$ in the label of an individual means that the individual belongs to the concept $\neg C$. Therefore, the node `TOP1` explicitly has `b` in its label and implicitly $\neg$`a`. A similar explanation holds for the node `TOP2`. An Alloy specification for the given example is presented below:

```
abstract sig TOP {
/* atomic roles */
    r : set TOP
}

sig BOTTOM in TOP {}
fact { #BOTTOM = 0 }

/* atomic concepts */

sig ifUnsatisfiableThenSubsumes in TOP {}
sig a in TOP {}  fact { #a > 0 }
sig b in TOP {}  fact { #b > 0 }
```

```
/* no individuals were declared */

/* equality axioms */

fact { (ifUnsatisfiableThenSubsumes) =
          (((( r.a ) & ( r.b )) &  (univ - (r.( a & b ))))) }

/* no inclusion axioms were declared */

/* no concept assertions were declared */

/* no role assertions were declared */

pred show() {
   #ifUnsatisfiableThenSubsumes = 1 and
   #univ = 3 and #a.r = 0 and #b.r = 0
 }

run show for 3
```

### 8.3.3 Counterexamples for a Concept Equivalence Assumption

As a next example we assume the following simple TBox:

$$\begin{aligned} \mathit{dogholder} &\equiv (\mathit{person} \sqcap (\geq 1\ \mathit{hasdog}.\mathit{dog})) \\ \mathit{houndholder} &\equiv (\mathit{person} \sqcap (\geq 1\ \mathit{hashound}.\mathit{hound})) \\ \mathit{dog} &\equiv \mathit{hound} \\ \mathit{hashound} &\sqsubseteq \mathit{hasdog} \end{aligned}$$

Suppose we expect that concepts *dogholder* and *houndholder* must be equivalent. In order to check this we let Alloy Analyzer generate models of the concept *counterExample*:

$$\mathit{counterExample} \equiv (\mathit{dogholder} \sqcap \neg \mathit{houndholder}) \sqcup (\neg \mathit{dogholder} \sqcap \mathit{houndholder})$$

One model found by Alloy Analyzer as a counterexample is shown in Figure 12. In this model, the individual `TOP3` is found that belongs to the concept `dogholder` but not to the concept `houndholder`. The reason is that the roles `hasdog` and `hashound` are not equivalent.


Figure 12: Model of *counterExample*

A specification in Alloy is given below:

```
abstract sig TOP {
/* atomic roles */
    hasdog : set TOP ,
```

```
    hashound : set TOP
}

sig BOTTOM in TOP {}
fact { #BOTTOM = 0 }

/* atomic concepts */
sig houndholder in TOP {}
sig counterExample in TOP {}
sig person in TOP {}
sig dogholder in TOP {}
sig hound in TOP {}
sig dog in TOP {}

/* equality axioms */
fact {  dog = hound }
fact {  hashound in hasdog }
fact { (houndholder) = ((person & ({ a : univ | #(a.(hashound :> hound)) >= 1 }))) }
fact { (dogholder) = (( person & ({ a : univ | #(a.(hasdog :> dog))   >= 1 }))) }
fact { (counterExample) = (((dogholder & (univ - houndholder))
                          + ((univ - dogholder) & houndholder))) }
pred show() {
#counterExample > 0
}
run show for 4
```

## 8.4 Evaluation of Practical Usefulness

In order to empirically study the performance of model generation with Alloy we have chosen a benchmark originally proposed for comparing DL systems (DL-98 systems comparison). We consider, k-branch, which evaluates satisfiability-testing performance for large concept expressions without reference to a TBox. Progressively larger expressions (from size 1 to size 21) are presented in two variants: all k-branch-p expressions are unsatisfiable while all k-branch-n expressions are satisfiable. These (and other) benchmarks are available at `http://dl.kr.org/dl98/comparison/data.html`. We also used RacerPro 1.9.1 beta to measure the times for (un)satisfiability checking with a tableau prover.

Summing up, Alloy exhibits a competitive performance for satisfiable input concepts if models can be found with up to 10 individuals. If models have more than 10 objects, performance quickly degrades (in particular, if unsatisfiable concepts are used as input). Apparently, BDD optimizations used in these systems cannot cope with combinatorial explosion, as more models are explored by Alloy than by tableau-based algorithms. Thus, there is good news when models are small enough (as full information can be presented to the ontology designer). If large model structures have to be explored, we found that model generators such as Alloy are not applicable.

As explained in the Alloy literature, the guiding principle for their construction was the "small scope hypothesis", which k-branch does not exhibit. Had we chosen a benchmark where this is the case, the results would have been more favorable to Alloy. For comparison, the time spent by RacerPro in this problem for different problem sizes is also shown in Table 5 and Table 6. The results of Alloy's runs are shown for different scope

sizes (we did not yet implemented an algorithm to compute the scope size according to the maximum concept branching factor and maximum depth of nested existential quantifiers as mentioned in Sect. 8.2).

As we can conclude from the measurement results, modern high-optimized tableau-based provers such as RacerPro far outperform model finders like Alloy, and therefore there is no reason to switch from one system to another completely. However, in order to improve the usefulness of tableau-based reasoners also for ontology design tasks, it may be a good idea to equip them with model generation capacities like those provided by model finders.

## 8.5 Conclusion and Further Work

In this work, we have studied an applicability of finite model finders (by the example of Alloy Analyzer) for ontology design tasks. For this, we have proposed translation rules from DL ($\mathcal{ALC}$ and some constructs of $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$) into Alloy and illustrated the translation by several examples. We conducted experiments that demonstrate the benefit model finders but also indicate their weaknesses.

While originally addressing interactive systems, in particular communication protocols, model-checking techniques are now applied to general imperative algorithms, as exemplified by the $^+$CAL algorithm language [Lam06]. Given that $^+$CAL allows specifying pre- and postconditions alongside imperative statements, it constitutes a viable mechanism for automatically testing the correctness of a proposed algorithm. A setting where this can be seen at work is the validation of the optimized implementation of a decision procedure. Indeed, model checkers might also be successfully applied for developing robust and scalable optimized description logics systems. Unlike testing, checking a specification even for a finite set of individual using model checking techniques might dramatically reduce development efforts.

The crucial requirement for integrating model finders in practical applications like ontology development tools is the effectiveness of constraint-solving engines they are based on. One of the recent investigations in producing high-performance tools is a Kodkod, a prototype SAT-based model finder designed for a relational logic [TJ06]. Besides of promising performance results, the system provides for further relevant features like optimized handling of assertional knowledge (in Alloy, specifying partial solutions is possible only in the form of additional constraints that increases the complexity of the solving process).

| Problem size | Alloy, 10 inds | Alloy, 15 inds | RacerPro |
|---|---|---|---|
| 1 | 265 | 110 | 3 |
| 2 | 110 | 328 | 5 |
| 3 | 1,797, NMF | 5,281 | 24 |
| 4 | 1,422, NMF | 21,921, NMF | 31 |
| 5 | 2,562, NMF | 43,687, NMF | 164 |
| 6 | 1,469, NMF | 31,125, NMF | 288 |
| 7 | 6,828, NMF | 61,625, NMF | 681 |
| 8 | 6,906, NMF | 42,625, NMF | 1,809 |
| 9 | 6,250, NMF | 53,000, NMF | 4,392 |
| 10 | 7,375, NMF | 4,970,229, NMF | 9,714 |
| 11 | 7437, NMF | 3,024,688, NMF | 23,623 |
| 12 | 17,50, NMF | 575,407, NMF | 51,266 |
| 13 | 27,640, NMF | 4,215,123, NMF | 119,628 |
| 14 | 4,281, NMF | 3,654,211, NMF | 294,519 |
| 15 | 38,577, NMF | 1,282,483, NMF | 765,325 |

Table 5: Concept satisfiability benchmarks (k-branch-n, all times in milliseconds, NMF = no model found).

| Problem size | Alloy, 10 inds | Alloy, 15 inds | RacerPro |
|---|---|---|---|
| 1 | 47 | 94 | 1 |
| 2 | 1,532 | 531 | 2 |
| 3 | 875 | 44,624 | 4 |
| 4 | 1,281 | 34,421 | 5 |
| 5 | 2,610 | 30,953 | 11 |
| 6 | 9,828 | 56,422 | 24 |
| 7 | 5,781 | 63,935 | 29 |
| 8 | 1,984 | 41,578 | 218 |
| 9 | 6,578 | 70,466 | 113 |
| 10 | 58,718 | 716,200 | 225 |
| 11 | 12,500 | 520,077 | 638 |
| 12 | 30,484 | 345,288 | 711 |
| 13 | 6,500 | 409,849 | 1,099 |
| 14 | 10,624 | 811,636 | 3,517 |
| 15 | 11,719 | 3,129,982 | 4,143 |
| 16 | 5,066 | 845,979 | 11,742 |
| 17 | 7,219 | 1,204,383 | 24,594 |
| 18 | 94,466 | 1,401,839 | 31,498 |
| 19 | 8,141 | 660,968 | 63,331 |
| 20 | 15,090 | 2,096,752 | 187,332 |
| 21 | 135,829 | 563,153 | 197,030 |

Table 6: Concept unsatisfiability benchmarks (k-branch-p, all times in milliseconds)

# 9 Conservative Extensions

## 9.1 Introduction

The paper [GLW06], which presents work that has been carried out within the TONES project, introduces conservative extensions in the context of description logics and identifies them as a central notion for the iterative development of ontologies and for ontology merging. Formally, conservative extensions are formulated as follows. For a TBox $\mathcal{T}$, the *signature* of $\mathcal{T}$ (denoted $\mathsf{sig}(\mathcal{T})$ is the set of all concept and role names used in $\mathcal{T}$. Then, a TBox $\mathcal{T}_1 \cup \mathcal{T}_2$ is a *conservative extension* of the TBox $\mathcal{T}_1$ if, for all concepts $C, D$ formulated in $\mathsf{sig}(\mathcal{T}_1)$, we have that $C \sqsubseteq_{\mathcal{T}_1 \cup \mathcal{T}_2} D$ implies $C \sqsubseteq_{\mathcal{T}_1} D$.

In the context of ontology design, conservative extensions can be used in the following way. Suppose that an ontology designer has already developed and tested the core part of an ontology (i.e., TBox) $\mathcal{T}$ that formalizes an application domain. Assume that he wants to extend $\mathcal{T}$ with a number of additional axioms that describe the terminology of a part of the domain that was not yet covered by $\mathcal{T}$. Then, it is important that the extension of $\mathcal{T}$ does not compromise the existing part of $\mathcal{T}$. In particular, the extended ontology should not entail new subsumptions between concepts that are formulated in the signature of the old ontology. By deciding whether the extended ontology is a conservative extension of the original one, we can check whether such subsumptions exist. Thus, conservative extensions are a crucial ingredient to the ontology design task of stepwise extension, where the idea is that the designer repeatedly extends the ontology with notions from additional parts of the application domain. Note that conservative extensions also play a core role in the definition of modules in an ontology, see [GHKS07]. Regarding the comon logical framework established in Deliverable D08, it is important to note that conservative extensions are concerned with stand-alone ontologies.

In this section, we summarize the results obtained in the papers [GLW06] and [LWW07]. In Section 9.3, we show that deciding conservative extensions in the basic description logic $\mathcal{ALC}$ is decidable and pinpoint it's exact computational complexity. In Section 9.4, we then consider a natural model-theoretic variation of the notion of a conservative extension and show that deciding this variation is highly undecidable already for $\mathcal{ALC}$. In Section 9.5, we consider the more expressive DL $\mathcal{ALCQI}$ and show that conservative extensions (in the original sense) are still decidable and that the complexity is the same as for $\mathcal{ALC}$. Finally, Section 9.6 contains a proofs that conservative extensions in $\mathcal{ALCQIO}$ are undecidable. It is important to note that the latter result implies undecidability of conservative extensions in the ontology language OWL.

## 9.2 Preliminaries

It is not difficult to see that $\mathcal{T}_1 \cup \mathcal{T}_2$ is a conservative extension of $\mathcal{T}_1$ iff there exists a concept $C$ such that $C$ is satisfiable w.r.t. $\mathcal{T}_1$, but not w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_2$. We call such a concept a *witness concept*. Thus, deciding whether an extension is conservative amounts to deciding the existence of a witness concept.

For some applications, it is more natural to consider the following variant of conservative extensions. Let $\Gamma$ be a signature, i.e., a finite set of concept and role names. Let

$\mathcal{T}_1$ and $\mathcal{T}_2$ be TBoxes. Then, $\mathcal{T}_1 \cup \mathcal{T}_2$ is *a conservative extension of* $\mathcal{T}_1$ *w.r.t.* $\Gamma$ if, for all $C_1, C_2 \in \mathcal{ALC}(\Gamma)$, we have $\mathcal{T}_1 \models C_1 \sqsubseteq C_2$ iff $\mathcal{T}_1 \cup \mathcal{T}_2 \models C_1 \sqsubseteq C_2$. This version of conservative extensions allows more control by specifying a signature $\Gamma$. In particular, the user can select a signature $\Gamma$ that he does not want to be compromised by an extension, instead of always using $\mathsf{sig}(\mathcal{T}_1)$ as in the original version of conservative extensions.

The two kinds of conservative extensions give rise to two reasoning problems:

- *Deciding conservative extensions* means, given TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$, to decide whether $\mathcal{T}_1 \cup \mathcal{T}_2$ is a conservative extension of $\mathcal{T}_1$
- *Deciding relativized conservative extensions* means, given TBoxes $\mathcal{T}_1$, $\mathcal{T}_2$ and a signature $\Gamma \subseteq \mathsf{sig}(\mathcal{T}_1)$, to decide whether $\mathcal{T}_1 \cup \mathcal{T}_2$ is a conservative extension of $\mathcal{T}_1$ w.r.t. $\Gamma$.

## 9.3 Conservative Extensions in $\mathcal{ALC}$

We start with stating a basic result about the complexity of deciding conservative extensions in $\mathcal{ALC}$ and an upper bound on the length of witness concepts. In the following, we denote by $|C|$ the length of a concept $C$. Similarly, the *size* $|\mathcal{T}|$ of a TBox $\mathcal{T}$ is defined as $\sum_{C \sqsubseteq D \in \mathcal{T}}(|C| + |D|)$.

**Theorem 9.1** *It is 2*ExpTime*-complete to decide relativized and non-relativized conservative extensions in $\mathcal{ALC}$. In both cases, if input TBoxes are $\mathcal{T}_1$ and $\mathcal{T}_2$ and the extension is not conservative, then there exists a witness concept $C$ of length at most 3-exponential in $|\mathcal{T}_1 \cup \mathcal{T}_2|$ that can be computed in time polynomial in $|C|$.*

As we will see later, the upper bound on the length of witness concepts given in Theorem 9.1 is tight. We now refine Theorem 9.1 by distinguishing between the size of the TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$. Such a more fine-grained analysis is rewarding if the sizes of $\mathcal{T}_1$ and $\mathcal{T}_2$ can be expected to differ substantially. This will usually be the case if an existing TBox is extended with a set of new GCIs: then, $|\mathcal{T}_2|$ is small compared to $|\mathcal{T}_1|$. In contrast, when merging two existing TBoxes, then no obvious assumption can be made concerning the relative size of $\mathcal{T}_1$ and $\mathcal{T}_2$.

It turns out that, when discriminating the size of $\mathcal{T}_1$ and $\mathcal{T}_2$, a difference in computational complexity emerges between deciding non-relativized and relativized conservative extensions. We first consider the former and show that there exists a decision procedure that is only exponential in the size of $\mathcal{T}_1$, but double exponential in the size of $\mathcal{T}_2$. We cannot expect a better bound in the size of $\mathcal{T}_1$: it follows from an easy reduction of satisfiability of $\mathcal{ALC}$ concepts w.r.t. TBoxes that deciding conservative extensions is ExpTime-hard even if $\mathcal{T}_2$ is assumed to be constant. We also provide a refined upper bound for the length of witness concepts by proving that, if $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$, then one can compute witness concepts of size 'only' 2-exponential in the size of $\mathcal{T}_1$, but 3-exponential in the size of $\mathcal{T}_2$. A matching lower bound on the size of witness concepts is established as well.

**Theorem 9.2**

*(i) There exists a deterministic algorithm for deciding conservative extensions in $\mathcal{ALC}$ whose runtime is bounded by $2^{p(|\mathcal{T}_1|)\times 2^{p(|\mathcal{T}_2|)}}$, with $p$ a polynomial.*

*(ii) For all $\mathcal{ALC}$ TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$, if $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$, then there exists a witness concept of length at most $2^{2^{|\mathcal{T}_1|\times 2^{|\mathcal{T}_2|}}}$.*

*(iii) There exist families of $\mathcal{ALC}$ TBoxes $(\mathcal{T}_n)_{n>0}$ and $(\mathcal{T}'_n)_{n>0}$, such that, for all $n > 0$,*

- *$\mathcal{T}_n \cup \mathcal{T}'_n$ is not a conservative extension of $\mathcal{T}_n$,*
- *$|\mathcal{T}_n| \in \mathcal{O}(n^2)$, $|\mathcal{T}'_n| \in \mathcal{O}(n^2)$, and*
- *every witness concept for $(\mathcal{T}_n, \mathcal{T}'_n)$ is of length at least $2^{(2^n \times 2^{2^n})-1}$.*

A proof of this theorem can be found in [GLW06]. The algorithm underlying Point (i) bears some similarity to the algorithm presented in Section 9.5.

We now refine our analysis of relativized conservative extensions. In contrast to the previous case, we can prove that there exists no decision procedure that is only single exponential in the size of $\mathcal{T}_1$: even for a fixed TBox $\mathcal{T}_2$, the complexity of deciding whether $\mathcal{T}_1 \cup \mathcal{T}_2$ is a conservative extension of $\mathcal{T}_1$ w.r.t. $\Gamma$ is 2ExpTime-hard. The proof is based on the following reduction of non-relativized conservative extensions to relativized conservativity w.r.t. a fixed extension.

**Lemma 9.3** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be $\mathcal{ALC}$ TBoxes, $\Gamma = \mathsf{sig}(\mathcal{T}_1)$, and $B$ a concept name not used in $\mathcal{T}_1$ and $\mathcal{T}_2$. Denote by $\mathcal{T}_2^B$ the TBox resulting from $\mathcal{T}_2$ by replacing every implication $C_1 \sqsubseteq C_2 \in \mathcal{T}_2$ with $C_1 \sqcap B \sqsubseteq C_2$. Let $\mathcal{T}_3 = \{B = \top\}$. Then the following are equivalent, for every $C \in \mathcal{ALC}(\Gamma)$:*

- *$C$ is satisfiable w.r.t. $\mathcal{T}_1$ but not w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_2$;*
- *$C$ is satisfiable w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_2^B$ but not w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_2^B \cup \mathcal{T}_3$.*

*It follows that $\mathcal{T}_1 \cup \mathcal{T}_2$ is a conservative extension of $\mathcal{T}_1$ iff $\mathcal{T}_1 \cup \mathcal{T}_2^B \cup \mathcal{T}_3$ is a conservative extension of $\mathcal{T}_1 \cup \mathcal{T}_2^B$ w.r.t. $\Gamma$.*

Together with Theorem 9.2, Lemma 9.3 allows us to establish the desired 2ExpTime lower bound. A matching upper bound is obtained from Theorem 9.1. Using Theorem 9.2 and Lemma 9.3, we can also establish a lower bound on the size of witness concepts that is *triple* exponential in the size of $\mathcal{T}_1$. This should be contrasted with the upper bound on witness concepts given in Theorem 9.2 for the case of non-relativized conservative extensions, which is only double exponential in $\mathcal{T}_1$.

**Theorem 9.4** *Let $\mathcal{T}'$ be an $\mathcal{ALC}$ TBox of the form $\{B = \top\}$, with $B$ a concept name. Then*

*(i) it is 2ExpTime-complete to decide, given an $\mathcal{ALC}$ TBox $\mathcal{T}_1$ and a signature $\Gamma \subseteq \mathsf{sig}(\mathcal{T}_1)$, whether $\mathcal{T}_1 \cup \mathcal{T}'$ is a conservative extension of $\mathcal{T}_1$ w.r.t. $\Gamma$.*

*(ii) there exist families of $\mathcal{ALC}$ TBoxes $(\mathcal{T}_n)_{n>0}$ and signatures $(\Gamma_n)_{n>0}$, such that, for all $n > 0$,*

- *$\mathcal{T}_n \cup \mathcal{T}'$ is not a conservative extension of $\mathcal{T}_n$ w.r.t. $\Gamma_n$,*
- *$|\mathcal{T}_n| \in \mathcal{O}(n^2)$, and*
- *every witness concept for $(\mathcal{T}_n, \mathcal{T}', \Gamma_n)$ is of length at least $2^{(2^n \cdot 2^{2^n})-1}$.*

Again, a proof can be found in [GLW06].

## 9.4 Model-Conservative Extensions in $\mathcal{ALC}$

In mathematical logic and software specification [Mai97], there are two different kinds of conservative extensions: one that is based on the consequence relation "$\models$" and one that is based on models only . For simplicity, we formulate this second notion only for non-relativized conservative extensions.

**Definition 9.5** [Model Conservative Extension] Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be TBoxes. We say that $\mathcal{T}_1 \cup \mathcal{T}_2$ is a *model conservative extension* of $\mathcal{T}_1$ iff for every model $\mathcal{I}$ of $\mathcal{T}_1$, there exists a model of $\mathcal{T}_1 \cup \mathcal{T}_2$ which can be obtained from $\mathcal{I}$ by modifying the interpretation of the predicates in $\mathsf{sig}(\mathcal{T}_2) \setminus \mathsf{sig}(\mathcal{T}_1)$ while leaving the predicates in $\mathsf{sig}(\mathcal{T}_1)$ fixed.

To distinguish the two versions of conservative extensions, in this section we call the original one *deductive conservative extension*.

The notion of a model conservative extension is more strict than the deductive one: if $\mathcal{T}_1 \cup \mathcal{T}_2$ is a model conservative extension of $\mathcal{T}_1$, then it is clearly also a deductive conservative extension of $\mathcal{T}_1$, but the converse does not hold. To see the latter, consider the TBoxes

$$\mathcal{T}_1 = \{\exists r.\top \sqcap \exists s.\top = \top\}, \quad \mathcal{T}_2 = \{\exists r.A \sqcap \exists s.\neg A = \top\}.$$

It is not hard to see that $\mathcal{T}_1 \cup \mathcal{T}_2$ is a deductive conservative extension of $\mathcal{T}_1$ if $\mathcal{ALC}$ (or even $\mathcal{ALCQI}$) is the language for witness concepts, but it is not a model conservative extension. The stronger notion of model conservative extensions is of interest for *query answering modulo ontologies*. In this case, one might want to ensure that under the addition of any ABox $\mathcal{A}$ (over the signature of $\mathcal{T}_1$) the answers to queries (over the signature of $\mathcal{T}_1$) to $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{A}$ coincide with those to $\mathcal{T}_1 \cup \mathcal{A}$. This immedialy follows if $\mathcal{T}_1 \cup \mathcal{T}_2$ is a model conservative extension of $\mathcal{T}_1$, but it does not follow if $\mathcal{T}_1 \cup \mathcal{T}_2$ is just a deductive conservative extension of $\mathcal{T}_1$.

However, from an algorithmic viewpoint model conservative extensions are a problematic choice: we show that they are highly undecidable even in the basic description logic $\mathcal{ALC}$ (and therefore also in all its extensions). The proof of the following result is by a reduction from the semantic consequence problem in modal logic.

**Theorem 9.6** *It is $\Pi^1_1$-hard to decide whether for two given $\mathcal{ALC}$ TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$, the TBox $\mathcal{T}_1 \cup \mathcal{T}_2$ is a model conservative extension of $\mathcal{T}_1$.*

## 9.5 Conservative Extensions in $\mathcal{ALCQI}$

We give a tight complexity bound for deciding conservative extensions in $\mathcal{ALCQI}$.

**Theorem 9.7** *It is 2-*ExpTime*-complete to decide conservative extensions in $\mathcal{ALCQI}$. In the case that $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$, there exists a witness concept $C$ of length at most 3-exponential in $|\mathcal{T}_1 \cup \mathcal{T}_2|$ that can be computed in time polynomial in $|C|$.*

The lower bound can be proved exactly in the same way as the 2-ExpTime lower bound for conservative extensions in $\mathcal{ALC}$. However, the lower bounds from $\mathcal{ALC}$ do *not simply transfer* to $\mathcal{ALCQI}$ and it is necessary to walk through the proof and check that it also works for the case of $\mathcal{ALCQI}$. In the following, we concentrate on proving the upper bound. It is established by devising a 2-ExpTime algorithm that, for convenience, decides *non*-conservative extensions.

We start by reminding that $\mathcal{ALCQI}$ has the tree model property. More precisely, a *tree interpretation* is an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, <^{\mathcal{I}})$ equipped with an additional relation $<^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that (i) $(\Delta^{\mathcal{I}}, <^{\mathcal{I}})$ is a tree, (ii) $\bigcup_{r \in \mathsf{N_R}} r^{\mathcal{I}} \cup r^{-\mathcal{I}} = < \cup <^{-1}$, and (iii) $s^{\mathcal{I}}$ and $r^{\mathcal{I}}$ are disjoint for all distinct roles $s$ and $r$. In $\mathcal{ALCQI}$, every concept $C$ that is satisfiable w.r.t. a TBox $\mathcal{T}$ is satisfiable in a *tree model* of $\mathcal{T}$, i.e., a model of $\mathcal{T}$ that is a tree interpretation [HST99]. In this section, when talking of an interpretation or model of a TBox we always mean a tree interpretation.

To develop the algorithm for deciding non-conservative extensions in $\mathcal{ALCQI}$, we introduce a new kind of witness for non-conservativity. The new witnesses are very similar to finite tree interpretations and easier to work with than witness concepts. For a signature $\Gamma$, let a *literal type* $S$ for $\Gamma$ be a subset of $\mathsf{lit}(\Gamma) := \{A, \neg A \mid A \in \Gamma \cap \mathsf{N_C}\}$ such that for each $A \in \Gamma \cap \mathsf{N_C}$, $A \in S$ iff $\neg A \notin S$. A $\Gamma$-*role* is a role $r$ such that $r$ or $r^-$ is in $\Gamma$.

**Definition 9.8** [$\Gamma$-tree] A $\Gamma$-*tree* $\mathfrak{T} = (W, <, L, O)$ is a finite intransitive tree $(W, <)$ such that each node $w \in W$ is labeled by a literal type $L(w)$ for $\Gamma$, each edge $(w, w')$ is labeled by a $\Gamma$-role $L(w, w')$, and $O \subseteq W$ is a set of leafs of $(W, <)$.

Essentially, a $\Gamma$-tree is a finite tree interpretation equipped with an additional unary predicate $O$ denoting a subset of the leafs. The following definition provides a way to relate $\Gamma$-trees and actual interpretations.

**Definition 9.9** [$\Gamma$-embedding] Let $\mathfrak{T} = (W, <, L, O)$ be a $\Gamma$-tree with root $w \in W$, and $\mathcal{I}$ an interpretation with root $d \in \Delta^{\mathcal{I}}$. A $\Gamma$-embedding $f : \mathfrak{T} \to \mathcal{I}$ is an injection from $W$ to $\Delta^{\mathcal{I}}$ such that

- $f(w) = d$,
- $L(v, v') = r$ iff $f(v) r^{\mathcal{I}} f(v')$, for all $v, v' \in W$ and $\Gamma$-roles $r$,
- $C \in L(v)$ iff $f(v) \in C^{\mathcal{I}}$, for all $v \in W$ and $C \in \mathsf{lit}(\Gamma)$,
- if $v \notin O$, then every $d' \in \Delta^{\mathcal{I}}$ with $f(v) r^{\mathcal{I}} d'$ for some $\Gamma$-role $r$ is in the range of $f$.

$\mathfrak{T}$ is called $\Gamma$-*embeddable into* $\mathcal{I}$ if there is a $\Gamma$-embedding $f : \mathfrak{T} \to \mathcal{I}$.

The definition illustrates that Γ-trees represent a (finite) initial part of (potentially infinite) tree interpretations. This explains the predicate $O$ of Γ-trees: $O$ marks those leafs in the Γ-tree that are not necessarily leafs in the tree interpretation $\mathcal{I}$ that we embed into. We can now establish Γ-trees as witnesses for non-conservativity.

**Lemma 9.10** *$\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$ w.r.t. Γ iff there exists a Γ-tree $\mathfrak{T} = (W, <, L, O)$ which is Γ-embeddable into a model of $\mathcal{T}_1$ but not into any model of $\mathcal{T}_1 \cup \mathcal{T}_2$.*

The general idea behind the algorithm is as follows: by Lemma 9.10, to decide whether $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$, it suffices to decide whether there exists a Γ-tree that is Γ-embeddable into a model of $\mathcal{T}_1$, but not into any model of $\mathcal{T}_1 \cup \mathcal{T}_2$. This is what our algorithm will do. Alas, we conjecture that there are cases in which the smallest such tree is 3-exponential in $|\mathcal{T}_1 \cup \mathcal{T}_2|$, and therefore a 2-exponential algorithm cannot simply try to construct such a tree. Instead, we check the existence of the Γ-tree by searching for certain witnesses for the existence of such a tree. Before we can introduce these witnesses (which should not be confused with Γ-trees as witnesses for non-conservativity), we need to introduce the notion of a type.

**Definition 9.11** [Type] Let $\mathcal{T}$ be a TBox. We use $\mathsf{cl}(\mathcal{T})$ to denote the smallest set that contains all concepts in $\mathcal{T}$ and is closed under single negations and under subconcepts. A *$\mathcal{T}$-type* $t$ is a subset of $\mathsf{cl}(\mathcal{T})$ such that

- $\neg C \in t$ iff $C \notin t$, for all $\neg C \in \mathsf{cl}(\mathcal{T})$;
- $C_1 \sqcap C_2 \in t$ iff $C_1 \in t$ and $C_2 \in t$, for all $C_1 \sqcap C_2 \in \mathsf{cl}(\mathcal{T})$.

Given an interpretation $\mathcal{I}$ and $u \in \Delta^{\mathcal{I}}$, the set

$$t^{\mathcal{I}}_{\mathcal{I}}(u) = \{C \in \mathsf{cl}(\mathcal{T}) \mid u \in C^{\mathcal{I}}\}$$

is a $\mathcal{T}$-type. In what follows, we will not always distinguish between the type $t$ and the conjunction of all members of $t$. We now introduce a witness for the existence of a Γ-tree that is Γ-embeddable into a model of $\mathcal{T}_1$, but not into any model of $\mathcal{T}_1 \cup \mathcal{T}_2$. To avoid writing sub- and superscripts, from now on we assume the input $\mathcal{T}_1$, $\mathcal{T}_2$, and Γ to be fixed.

**Definition 9.12** [Root pair, Internal pair] A *root pair* $(t, U)$ consists of a $\mathcal{T}_1$-type $t$ and a set $U$ of $\mathcal{T}_1 \cup \mathcal{T}_2$-types. An *internal pair* $(t' \rightarrow_r t, U)$ consists of a Γ-role $r$, $\mathcal{T}_1$-types $t'$ and $t$, and a function $U$ mapping each $\mathcal{T}_1 \cup \mathcal{T}_2$-type to a set of $\mathcal{T}_1 \cup \mathcal{T}_2$-types.

Intuitively, each (root or internal) pair encodes relevant information about possible embeddings of a Γ-tree into models of $\mathcal{T}_1$ and $\mathcal{T}_1 \cup \mathcal{T}_2$. This is made precise by the notion of realizability.

**Definition 9.13** [Realizable root pair] Let $\mathfrak{T} = (W, <, L, O)$ be a Γ-tree. A root pair $(t, U)$ is *realized* by $\mathfrak{T}$ iff

1. there exist a model $\mathcal{I}$ of $\mathcal{T}_1$ with root $d \in t^{\mathcal{I}}$ and a Γ-embedding $f : \mathfrak{T} \rightarrow \mathcal{I}$;
2. for every $\mathcal{T}_1 \cup \mathcal{T}_2$-type $s$, we have $s \in U$ iff there exist a model $\mathcal{I}$ of $\mathcal{T}_1 \cup \mathcal{T}_2$ with root $d \in s^{\mathcal{I}}$ and a Γ-embedding $f : \mathfrak{T} \rightarrow \mathcal{I}$.

While root pairs encode information about possible embeddings of a Γ-tree into models of $\mathcal{T}_1$ and $\mathcal{T}_1 \cup \mathcal{T}_2$, internal pairs encode information about possible embeddings of a Γ-tree into *rooted submodels* of models of $\mathcal{T}_1$ and $\mathcal{T}_1 \cup \mathcal{T}_2$. In the following, if $\mathcal{I}$ is a (tree) interpretation and $d \in \Delta^{\mathcal{I}}$, we write $\mathcal{I}_d$ to denote the sub-tree interpretation of $\mathcal{I}$ rooted at $d$.

**Definition 9.14** [Realizable internal pair] Let $\mathfrak{T} = (W, <, L, O)$ be a Γ-tree. An internal pair $(t' \rightarrow_r t, U)$ is realized by $\mathfrak{T}$ iff

- there exist a model $\mathcal{I}$ of $\mathcal{T}_1$ and $d', d \in \Delta^{\mathcal{I}}$ such that $d' \in (t')^{\mathcal{I}}$, $d' r^{\mathcal{I}} d$, $d \in t^{\mathcal{I}}$, and there is a Γ-embedding $f : \mathfrak{T} \rightarrow \mathcal{I}_d$;
- for all $\mathcal{T}_1 \cup \mathcal{T}_2$-types $s, s'$, we have $s' \in U(s)$ iff there exist a model $\mathcal{I}$ of $\mathcal{T}_1 \cup \mathcal{T}_2$ and $d', d \in \Delta^{\mathcal{I}}$ such that $d' \in (s')^{\mathcal{I}}$, $d' r^{\mathcal{I}} d$, $d \in s^{\mathcal{I}}$, and there is a Γ-embedding $f : \mathfrak{T} \rightarrow \mathcal{I}_d$.

A (root or internal) pair is *realizable* if there exists a Γ-tree $\mathfrak{T}$ which realizes it.

Observe that internal pairs store information not only about the element $d \in \Delta^{\mathcal{I}}$ to which the root of $\mathfrak{T}$ is mapped, but also comprise the type $t'$ of the predecessor $d'$ of $d$ in $\mathcal{I}$ and the (unique!) role $r$ which connects $d'$ and $d$. This is necessary due to the presence of inverse roles and number restrictions and bears some similarity to the *double blocking* technique in tableau algorithms; see [HST99]. Also note that the $U$-component of internal pairs is a function rather than a set because, intuitively, the possible types of $d$ in models of $\mathcal{T}_1 \cup \mathcal{T}_2$ depend on the type of the predecessor $d'$ in such models.

Let us now describe the algorithm. By Lemma 9.10 and definition of realizability, there exists a realizable root pair of the form $(t, \emptyset)$ iff $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$ w.r.t. Γ. The algorithm for deciding non-conservative extensions searches for such a root pair. The easiest case is that a root pair $(t, \emptyset)$ is realized by a *singleton* Γ*-tree*, i.e., a Γ-tree that consists of only a single node. This special case is tested first. If the test is not successful, we must check whether there is a root pair $(t, \emptyset)$ that is realized by a non-singleton tree $\mathfrak{T} = (W, <, L, O)$. Assume that this is the case and that the root of $\mathfrak{T}$ is $w$. Then each subtree of $\mathfrak{T}$ rooted at a succesor node $w'$ of $w$ realizes an internal pair $(\hat{t}' \rightarrow_{\hat{r}} \hat{t}, \hat{U})$ with $\hat{t}' = t$ and $\hat{r} = L(w, w')$. Intuitively, this means that we can check realization of the root pair $(t, \emptyset)$ in $\mathfrak{T}$ based on the realization of internal pairs in trees of strictly smaller height. Similarly, we can check the realizability of *internal* pairs in a Γ-tree based on the realizability of internal pairs in Γ-trees of strictly smaller height. Based on these observations, our algorithm repeatedly generates internal pairs that are realized by Γ-trees of larger and larger height until all such pairs are generated. It then checks whether there exists a root pair $(t, \emptyset)$ that is realizable based on the generated internal pairs. The following definition formalizes one step of the algorithm in which root pairs or new internal pairs are generated from an existing set of internal pairs.

In the following, if $\mathfrak{T}$ is a Γ-tree and $w \in W$, we write $\mathfrak{T}_w$ to denote the sub-tree of $\mathfrak{T}$ rooted at $w$.

**Definition 9.15** [One step] Let $\mathcal{R}$ be a set of internal pairs. A root pair $(t, U)$ (resp. internal pair $(t' \rightarrow_r t, U)$) can be *obtained in one step* from $\mathcal{R}$ if there exists a Γ-tree $\mathfrak{T} = (W, <, L, O)$ with root $w$ such that

Suppose TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$, and a signature $\Gamma \subseteq sig(\mathcal{T}_1)$ are given.

1. Determine the set $\mathcal{R}_0^r$ of root pairs realized by singleton $\Gamma$-trees. If $\mathcal{R}_0^r = \emptyset$, then reject the input (because $\mathcal{T}_1$ is not satisfied in any model). If $\mathcal{R}_0^r$ contains a root pair $(t, U)$ such that $U = \emptyset$, then accept. Else,

2. Determine the set $\mathcal{R}_0$ of internal pairs realizable by singleton $\Gamma$-trees. If $\mathcal{R}_0 = \emptyset$, then reject the input. Else,

3. Generate the sequence $\mathcal{R}_1, \mathcal{R}_2, \ldots$ of sets of internal pairs such that

$$\mathcal{R}_{i+1} = \mathcal{R}_i \cup \mathcal{R}_i',$$

where $\mathcal{R}_i'$ is the set internal pairs which can be obtained from some non-empty subset of $\mathcal{R}_i$ of cardinality not exceeding $m_{\mathcal{T}_1,\mathcal{T}_2}$ in one step. This is done until $\mathcal{R}_i = \mathcal{R}_i \cup \mathcal{R}_i'$. Then accept the input if there exists a root pair $(t, U)$ with $U = \emptyset$ which can be obtained in one step from some subset of $\mathcal{R}_i$ of cardinality not exceeding $m_{\mathcal{T}_1,\mathcal{T}_2}$. If no such root pair exists, reject the input.

Figure 13: Algorithm for non-conservativeness w.r.t. $\Gamma$.

- $\mathfrak{T}$ realizes $(t, U)$ (resp. $(t' \to_r t, U)$);
- for all $w' \in W$ with $w < w'$, there exists an internal pair $p = (\hat{t}' \to_{\hat{r}} \hat{t}, \hat{U}) \in \mathcal{R}$ such that $\hat{t}' = t$, $\hat{r} = L(w, w')$, and $p$ is realized by $\mathfrak{T}_{w'}$.

The details of our algorithm are given in Figure 13, where

$$m_{\mathcal{T}_1,\mathcal{T}_2} := 2 \times |\mathcal{T}_1 \cup \mathcal{T}_2| \times 2^{3 \times |\mathcal{T}_1 \cup \mathcal{T}_2|}.$$

Intuitively, considering only a subset of $\mathcal{R}_i$ of cardinality $m_{\mathcal{T}_1,\mathcal{T}_2}$ means that we limit our attention to $\Gamma$-trees of out-degree $m_{\mathcal{T}_1,\mathcal{T}_2}$. This is justified by the following lemma.

**Lemma 9.16** *If $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$ w.r.t. $\Gamma$, then there exists a root pair $(t, \emptyset)$ realized by a $\Gamma$-tree $\mathfrak{T}$ of outdegree at most $m_{\mathcal{T}_1,\mathcal{T}_2}$.*

It remains to be shown that each step of the algorithm can be carried out effectively and that the algorihm yields the 2-ExpTime upper bound stated in Theorem 9.7. We start with the former. The proof of the following lemma relies on the fact that satisfiability in $\mathcal{ALCQI}$ w.r.t. TBoxes can be decided in ExpTime [Tob01].

**Lemma 9.17** *It can be checked in 2-exponential time (in the size of $\mathcal{T}_1$, $\mathcal{T}_2$) whether a (root or internal) pair can be obtained in one step from a set $\mathcal{R}$ of realizable internal pairs with $|\mathcal{R}| \leq m_{\mathcal{T}_1,\mathcal{T}_2}$.*

The number of internal pairs is bounded double exponentially in the size of $|\mathcal{T}_1 \cup \mathcal{T}_2|$. Therefore, the third step of the algorithm stabilizes after at most double exponentially many rounds. Together with Lemma 9.17, it follows that our algorithm is a 2-ExpTime one.

**Theorem 9.18** *The algorithm in Figure 13 accepts input $\mathcal{T}_1$, $\mathcal{T}_2$, $\Gamma$ iff $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$ w.r.t. $\Gamma$. It runs in 2-exponential time.*

To show the upper bound on the size of witness concepts stated in Theorem 9.7, we proceed as follows: first, we observe that if the algorithm finds a realizable root pair $(t, \emptyset)$, then this pair is realized by a $\Gamma$-tree of at most double exponential depth and single exponential outdegree. Second, we show how to convert such a $\Gamma$-tree into a witness concept of three-exponential size.

## 9.6 Conservative Extensions in $\mathcal{ALCQIO}$

We show that conservative extensions are undecidable in $\mathcal{ALCQIO}$. The proof is by a reduction of the following undecidable tiling problem.

**Definition 9.19** A *domino system* $D = (T, H, V, R, L, T, B)$ consists of a finite set $T$ of *tiles*, horizontal and vertical *matching relations* $H, V \subseteq T \times T$, and sets $R, L, T, B \subseteq T$ of *right* tiles, *left* tiles, *top* tiles, and *bottom* tiles. A *solution* to $D$ is a triple $(n, m, \tau)$ where $n, m \geq 0$ and $\tau : \{0, \ldots, n-1\} \times \{0, \ldots, m-1\} \to T$ such that the following hold:

1. $(\tau(i, j), \tau(i+1, j)) \in H$, for all $i < n$ and $j \leq m$;
2. $(\tau(i, j), \tau(i, j+1)) \in V$, for all $i \leq n$ and $j < m$;
3. $\tau(0, j) \in L$ and $\tau(n, j) \in R$, for all $j \leq m$;
4. $\tau(i, 0) \in B$ and $\tau(i, m) \in T$, for all $i \leq n$.

Using proof methods from [vEB97], it is easy to show that it is undecidable whether a given domino system $D$ has a solution. We show how to convert a domino system $D$ into TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $D$ has a solution iff $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$. In particular, models of witness concepts will correspond to solutions of $D$.

Let $D = (T, H, V, R, L, T, B)$ be a domino system. The TBox $\mathcal{T}_1$ uses the following signature: an indidual name $o$, role names $r_x$ and $r_y$, concept names $\mathsf{top}$, $\mathsf{bottom}$, $\mathsf{left}$, and $\mathsf{right}$ and each element of $T$ as a concept name. The TBox $\mathcal{T}_1$ contains the following:

- The roles $r_x$, $r_y$, and their inverses are functional:

$$\top \sqsubseteq (\leqslant 1\ r\ \top), \text{ for } r \in \{r_x, r_y, r_x^-, r_y^-\}$$

- Every position in the $n \times m$ grid is labeled with exactly one tile and the matching conditions are satisfied:

$$\begin{aligned} \top &\sqsubseteq \bigsqcup_{t \in T} (t \sqcap \bigsqcap_{t' \in T,\, t' \neq t} \neg t') \\ \top &\sqsubseteq \bigsqcap_{t \in T} (t \to (\bigsqcup_{(t,t') \in H} \forall r_x.t' \sqcap \bigsqcup_{(t,t') \in V} \forall r_y.t')) \end{aligned}$$

- The concepts $\mathsf{left}$, $\mathsf{right}$, $\mathsf{top}$, $\mathsf{bottom}$ mark the boundaries of the grid in the expected way:

$$\begin{aligned} \mathsf{right} &\sqsubseteq \neg\exists r_x.\top \sqcap \forall r_y.\mathsf{right} \sqcap \forall r_y^{-1}.\mathsf{right} \\ \neg\mathsf{right} &\sqsubseteq \exists r_x.\top \end{aligned}$$

and similarly for $\mathsf{left}$, $\mathsf{top}$, and $\mathsf{bottom}$.

- The individual name $o$ marks the origin:

$$\{o\} \sqsubseteq \mathsf{left} \sqcap \mathsf{bottom}.$$

The TBox $\mathcal{T}_2$ introduces two new concept names $Q$ and $P$. It contains the following two concept inclusions:

$$\{o\} \sqsubseteq Q \sqsubseteq \exists r_x.Q \sqcup \exists r_y.Q \sqcup (\exists r_x.\exists r_y.P \sqcap \exists r_y \exists r_x.\neg P)$$

The idea behind this definition of $\mathcal{T}_2$ is to enforce that models $\mathcal{I}$ of witness concepts are such that (i) there is no infinite outgoing $r_x/r_y$-path starting at $o^{\mathcal{I}}$ and (ii) $r_x$ and $r_y$ commute in the connected part of $\mathcal{I}$ rooted at $o^{\mathcal{I}}$. This is achieved as follows: if (i) is violated, then we can find an assignment of $Q$ in $\mathcal{I}$ that satisfies $\mathcal{T}_2$. Similarly, if (ii) is violated, then we can find an assignment of ($Q$ and) $P$ in $\mathcal{I}$ that satisfies $\mathcal{T}_2$.

It can be checked that, as intended, $D$ has a solution iff $\mathcal{T}_1 \cup \mathcal{T}_2$ is not a conservative extension of $\mathcal{T}_1$. Here, we only show how to construct a witness concept in the case that $D$ has a solution. Such a witness concept $C$ has to ensure that for all models $\mathcal{I}$ of $C$ and $\mathcal{T}_1$, the connected part of $\mathcal{I}$ rooted at $o^{\mathcal{I}}$ is isomorphic to the $n \times m$-grid.

For every word $w \in \{r_x, r_y\}^*$, denote by $\overleftarrow{w}$ the word that is obtained by reversing $w$ and then adding $\cdot^-$ to each symbol. Let $|w|_r$ denote the number of occurrences of the symbol $r$ in $w$. Now, $C$ is the conjunction of

$$\{o\} \sqcap \forall r_x^n.\mathsf{right} \sqcap \forall r_y^m.\mathsf{top}$$

and for every $w \in \{r_x, r_y\}^*$ such that $|w|_{r_x} < n$ and $|w|_{r_y} < m$, the concept

$$\exists (w \cdot r_x r_y r_x^- r_y^- \cdot \overleftarrow{w}).\{o\},$$

where $\exists w.D$ abbreviates $\exists r_1.\cdots\exists r_k.D$ if $w = r_1 \cdots r_k$. It is readily checked that $C$ enforces an $n \times m$-grid as required.

**Theorem 9.20** *In $\mathcal{ALCQIO}$, conservative extensions are undecidable.*

Note that the theorem applies even to the case where $\Gamma = \mathsf{sig}(\mathcal{T}_1)$ and we allow $(\leqslant 1\ r\ \top)$ as the only form of number restriction.

## 9.7 Related Work

Conservative extensions in DLs to support ontology design and integration have only only been considered in the two papers that we have summarized in the preceeding sections and in the TONES publications [GHKS07]. In the latter, conservative extensions are used as a basis for different notions of modularity. Since the complexity of deciding conservative extensions is relatively high, it remains as important future work to understand whether the identified algorithms can be implemented in an efficient way to be useful in practice. If this is not the case, one may either try to identify more feasible algorithms or try to develop a more pragmatic approach. The syntactic restrictions presented in [GHKS07] are a promising step in this direction. It may also be interesting to analyze conservative extensions for other logics such as the dynamic epistemic logics analyzed within the TONES paper [Lut06].

# 10 FCA-based Completion of TBoxes

## 10.1 Introduction

In Section 2, DLs were introduced and it was mentioned that the DL-based ontology language OWL is standardized as the ontology language of the web. As a consequence of this standardization, many ontology editors now support OWL [BHGS01, KFNM04, OVSM04, KPS+05], and ontologies written in OWL are employed in more and more applications. As the size of such ontologies grows, tools that support improving the quality of large DL-based ontologies become more important. The tools available until now use DL reasoning to detect inconsistencies and to infer consequences. There are also first 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 [SC03b, Sch05a, Sch05b, PSK05, KPSCG06]. 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). Here, we are concerned with a different quality dimension: *completeness*. We want to develop 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. Given an application domain and a DL knowledge base (KB) describing it, we can ask whether the KB 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?

Obviously, completeness in this sense is a very relevant issue for structuring the ontology.

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 been identified?

Such questions cannot be answered by an automated tool alone. Clearly, to check whether a certain 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, new biological knowledge may be acquired by the expert in the process.

Attribute exploration [Gan84] is an approach developed in Formal Concept Analysis (FCA) [GW99] that can be used to acquire knowledge about an application domain by querying an expert. To answer a query whether a certain relationship holds, the expert must either confirm the relationship (by using results from the literature or providing 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 context, we cannot directly use this approach. The main reason is the open-world semantics of description logic knowledge bases. Consider an individual $i$ from the ABox and a concept $C$ occurring in the TBox. If we cannot deduce from the TBox and ABox 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 the TBox and ABox imply that $i$ is an instance of $\neg C$. Thus, our knowledge about the relationships between individuals and concepts is incomplete: if TBox and ABox 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 individuals is complete: the basic datastructure is that of a formal context, i.e., a crosstable between individuals and properties. A cross says that the property holds, and the absence of a cross is interpreted as saying that the property does not hold.

In the next section, we first briefly review some notions and results from FCA. Then, we develop our variant of FCA that can deal with partial contexts, and finally describe an attribute exploration procedure that works with partial contexts. In Section 10.5, we specialize our new attribute exploration procedure to the case of partial contexts induced by DL knowledge bases. For proofs of the results please see the technical report [BGSS06]. This work is going to be published as [BGSS07].

## 10.2 Formal Concept Analysis

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 is supposed to facilitate the use of mathematical reasoning for conceptual data analysis and knowledge processing. In FCA, one represents data in the form of a *formal context*, which in its simplest form is a way of specifying which attributes (properties) are satisfied by which objects (individuals). Formally, a formal context is defined as follows:

**Definition 10.1** 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 visualised as a crosstable, where the rows represent the objects, and the columns represent the attributes. 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 $m$. Here, we will always assume the set of attributes $M$ to be *finite*.

Let $\mathbb{K} = (G, M, I)$ be a formal context. For a set of objects $A \subseteq G$, the *intent* $A'$ of $A$ is the set of attributes that are satisfied by all objects in $A$, i.e.,

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

Similarly, for a set of attributes $B \subseteq M$, the *extent* $B'$ of $B$ is the set of objects that satisfy all attributes in $B$, i.e.,

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

It is easy to see that, for $A_1 \subseteq A_2 \subseteq G$ (resp. $B_1 \subseteq B_2 \subseteq M$), we have

- $A_2' \subseteq A_1'$ (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 10.2** 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 $A_1 \subseteq A_2$ (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 concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice in which infimum and supremum are given by:

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

Given a formal context, one common method to analyse it is to find (a base of) the implications between the 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 $m_{j1}, \ldots, m_{j\ell}$."

Formally, an implication between attributes is defined as follows:

**Definition 10.3** 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 implications that hold in $\mathbb{K}$ by $Imp(\mathbb{K})$.

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 10.4** 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 viewpoint of logic, computing the implication closure of a set of attributes is just computing consequences in propositional Horn logic. For this reason, the implicational 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, implications can be seen as functional dependencies in relational databases, and thus the linearity result can also be obtained by using methods for deriving new functional dependencies from given ones [Mai83].

**Definition 10.5** 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.

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 $B_2 \subseteq B_1'' = \mathcal{J}(B_1)$.

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 crosstable, but it is rather "known" to a domain expert. In such cases, Ganter's interactive knowledge acquisition method *attribute exploration* [Gan84] has proved to be a useful method to efficiently capture the expert's knowledge. By asking implication questions to a domain expert, the method computes a base for $Imp(\mathbb{K})$ and a subcontext $\mathbb{K}'$ of the $\mathbb{K}$ 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 the expert gives a counterexample from $\mathbb{K}$, which is then added to $\mathbb{K}'$.

In order to produce a base for $Imp(\mathbb{K})$, one could, of course, enumerate all possible implications, and have the expert decide for each of them whether it holds in $\mathbb{K}$ or not. Obviously, this would be very inefficient, and produce all of $Imp(\mathbb{K})$ rather than a small base for this set. The main idea underlying *attribute exploration* (see Algorithm 1) is that one can restrict the attention to implications having a left-hand side that is closed under

the implications of the context, and whose right-hand side is obtained from the left-hand side by applying the $\cdot''$ closure operator. The left-hand sides are enumerated in a certain order, called the *lectic order*, which ensures that it is sufficient to build the implication closure w.r.t. the already computed implications. In addition, the $\cdot''$ operator is computed w.r.t. the already computed subcontext rather than the full context $\mathbb{K}$.

**Definition 10.6** 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.,

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

Obviously, $<$ extends the strict subset order, and thus $\emptyset$ is the smallest and $M$ the largest set w.r.t. $<$.

The following proposition shows how one can enumerate all closed sets w.r.t. a given closure operator in the lectic order.

**Proposition 10.7** *Given a closure operator $\varphi$ on $M$ and a $\varphi$-closed set $A \subsetneq M$, the next $\varphi$-closed set following $A$ in the lectic order is*

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

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

Based on this observation, the attribute exploration algorithm is described in Algorithm 1 below. It can be shown that Algorithm 1 always terminates, and that the set of implications $\mathcal{L}_i$ obtained after termination is a base for $Imp(\mathbb{K})$. More precisely, one can show that it is the so-called *Duquenne-Guigues* base of the context, which contains a minimal number of implications. This base can be described independently of the algorithm, based on the notion of a *pseudo-intent* of the context.

**Definition 10.8** A set $P \subseteq M$ is called a pseudo-intent of the context $\mathbb{K} = (G, M, I)$ if $P \neq P''$ and $Q'' \subseteq P$ holds for all pseudo-intents $Q \subsetneq P$.

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

**Definition 10.9** The *Duquenne-Guigues* base of the context $\mathbb{K}$ consists of the implications $P \to P'' \setminus P$, where $P$ ranges over all pseudo-intents of $\mathbb{K}$.

**Algorithm 1** 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 $o$ of $\mathbb{K}$ from the expert s.t: $P_i \subseteq o'$ a nd $P_i'' \not\subseteq o'$
      $\mathbb{K}_{i+1} := \mathbb{K}_i \cup \{o\}$
      $P_{i+1} := P_i$
      $\mathcal{L}_{i+1} := \mathcal{L}_i$
    **end if**
  **else**
    $\mathbb{K}_{i+1} := \mathbb{K}_i$
    $\mathcal{L}_{i+1} := \mathcal{L}_i$
    $P_{i+1} := \mathcal{L}_{i+1}((P_i \cap \{m_1, \ldots, m_{j-1}\}) \cup \{m_j\})$ 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**

## 10.3 Partial contexts

The goal of this section is to extend the classical approach to FCA described above 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. As above, we assume that we have a finite set $M$ of attributes and a (possibly infinite) set of objects.

**Definition 10.10** 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 introduced in the previous section: a set of fods $\overline{\mathcal{K}}$ corresponds to the formal context $\mathbb{K}_{\overline{\mathcal{K}}} := (\overline{\mathcal{K}}, M, I)$, where $(\overline{A}, \overline{S}) I m$ 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 10.11** 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* $(A, S)$.

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

**Definition 10.12** 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 10.13** *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 $m \in \overline{\mathcal{K}}(P)$.

The following proposition connects refutation by a partial context to refutation by the realizers of this partial context.

**Proposition 10.14** *Let $\mathcal{K}$ be a partial context. An implication is refuted by $\mathcal{K}$ iff it is refuted by all realizers of $\mathcal{K}$.*

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 10.15** *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 $\mathcal{K}'$.*

## 10.4 Attribute exploration with partial contexts

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. $\overline{\mathcal{K}}$. [8] 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 as possible questions to the expert.

**Definition 10.16** 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.

[8] though finding these answers may involve literature study, or even proving new mathematical theorems or carrying out new experiments.

**Proposition 10.17** *Assume that $\mathcal{K} \leq \overline{\mathcal{K}} \subseteq Mod(\mathcal{L})$ and that all implications are decided w.r.t. $\mathcal{K}$ and $\mathcal{L}$. Then $\mathcal{L}$ is complete for $Imp(\overline{\mathcal{K}})$ and $\mathcal{K}$ refutes all implications not belonging to $Imp(\overline{\mathcal{K}})$.*

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 10.18** *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 10.13, 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 and the procedure derived from Proposition 10.7 for enumerating all $\mathcal{L}$-closed sets w.r.t. this order.

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. In reality, however, both $\mathcal{L}$ and $\mathcal{K}$ are changed 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 10.19** *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.*

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 10.20** *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 $\mathcal{K} \leq \mathcal{L}(\mathcal{K}) \leq \overline{\mathcal{K}}$.*

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 10.21** *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 $\mathcal{K}(L) = \mathcal{K}'(L)$.*

Based on these considerations, our attribute exploration algorithm for partial contexts is described in Algorithm 2. The following proposition shows that this algorithm always terminates, and in which sense it is correct.

**Proposition 10.22** *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 2 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 $\overline{\mathcal{K}}$.*

Next, we show that the set of implications $\mathcal{L}$ produced by the algorithm 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 10.23** *The set $\mathcal{L}$ computed by Algorithm 2 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 $Imp(\overline{\mathcal{K}})$.*

## 10.5 DLs 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

$$pod_{\mathcal{T},\mathcal{A}}(a, M) := (A, S) \text{ where } A := \{C \in M \mid \mathcal{T}, \mathcal{A} \models C(a)\} \text{ and } S := \{C \in M \mid \mathcal{T}, \mathcal{A} \models \neg C(a)\},$$

**Algorithm 2** 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**

and the whole ABox induces the partial context

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

Note that $pod_{\mathcal{T},\mathcal{A}}(a, M)$ 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

$$fod_{\mathcal{I}}(d, M) := (\overline{A}, \overline{S}) \text{ where } \overline{A} := \{C \in M \mid d \in C^{\mathcal{I}}\} \text{ and } \overline{S} := \{C \in M \mid d \in (\neg C)^{\mathcal{I}}\},$$

and the whole interpretation induces the full context

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

Note that $fod_{\mathcal{I}}(d, M)$ 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 10.24** *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 $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$.*

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

**Definition 10.25** 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 10.26** *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 $\mathcal{I}$.*

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 10.13. Since in the following the attribute set $M$ can be assumed to be fixed, we will write $\mathcal{K}_{\mathcal{T},\mathcal{A}}$ rather that $\mathcal{K}_{\mathcal{T},\mathcal{A}}(M)$. 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 10.13, $\mathcal{K}_{\mathcal{T},\mathcal{A}}(P)$ is the largest subset of $M$ such that $P \to \mathcal{K}_{\mathcal{T},\mathcal{A}}(P)$ is not refuted by $(\mathcal{T}, \mathcal{A})$.

## 10.6 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 10.27** 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 10.28** *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 the previous section 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 assertions) such that the extended ABox refutes $L \to R$ and $\mathcal{I}$ is still a model of this ABox. Because of Proposition 10.26, 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 3.

Note that Algorithm 3 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 2 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 10.22 immediately implies the following proposition.

**Proposition 10.29** *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 3 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 3 really computes a completion of the input knowledge base.

**Theorem 10.30** *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 3. Then $(\mathcal{T}, \mathcal{A})$ is a completion of $(\mathcal{T}_0, \mathcal{A}_0)$.*

## 10.7 Related Work

There has been some work on how to extend FCA and attribute exploration from complete knowledge to the case of partial knowledge. In [Obi02], a new three-valued modal logic for the evaluation of arbitrary propositional formulae under partial knowledge, and a deduction mechanism was presented. In [BH00, Hol04a, Hol04b, BH05], an attribute logic based on three-valued Kleene-logic and a version of attribute exploration for partial knowledge was introduced. However, this work is based on assumptions that are different from ours. In particular, it assumes 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 KB, 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.

**Algorithm 3** 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 whether 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**

# 11 Computing Common Subsumers

## 11.1 Introduction

By computing common subsumers of a set of concepts (possibly defined w.r.t. an ontology) one obtains a new concept that subsumes all concepts from the set. This new concept captures the information shared by all of the concepts from the set. So far mainly the task of computing *least common subsumers* (lcs) has been investigated [BKM99, KM01, Baa03a]. Intuitively, for a DL $\mathcal{L}$ the lcs of a set of concepts in $\mathcal{L}$ is the most specific $\mathcal{L}$-concept that subsumes $C_1, \ldots, C_n$.

However, the lcs is only useful in DLs that do *not* offer disjunction as a concept constructor: in the presence of disjunction, the lcs of concepts $C_1, \ldots, C_n$ is simply $C_1 \sqcup \cdots \sqcup C_n$. Clearly, the latter concept does not shed any light on the commonalities of $C_1, \ldots, C_n$.

One approach to obtain a "meaningful" common subsumer also for DLs that offer disjunction is proposed in [BKT02b] and extended in [BKT02a]. The idea is to first compute for each input concept the concept approximation, i.e., the closest concept (w.r.t. subsumption) in a DL that does not offer disjunction. Second, the lcs of these approximations is computed.

In more recent times, research has tried to overcome the main limitation of existing research in computing the lcs by employing a different framework, first proposed in [BST04a]. In this framework we 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 "user DL" and needs support through the bottom-up approach when building this user-specific extension of the background terminology–the user ontology. There are several reasons for the user to employing a restricted DL in this setting:

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

Thus the user DL does not offer concept disjunction. The lcs obtained for the selected concepts is then (edited by the user and) added to the user 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 and does not only build a disjunction of the input concepts. However, in addition to primitive concepts and roles, the concepts written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$. Let us call such concepts $\mathcal{L}_1(\mathcal{T})$-concepts. Given $\mathcal{L}_1(\mathcal{T})$-concepts $C_1, \ldots, C_n$, we are now looking for their lcs in $\mathcal{L}_1(\mathcal{T})$, i.e., the least $\mathcal{L}_1(\mathcal{T})$-concept that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. Depending

on the DLs $\mathcal{L}_1$ and $\mathcal{L}_2$, least common subsumers of $\mathcal{L}_1(\mathcal{T})$-concepts w.r.t. an $\mathcal{L}_2$-TBox $\mathcal{T}$ may or may not exist.

In the context of the TONES project this direction of research was further pursued, e.g. by extending the results given in [BST04a] to more expressive ontology languages in [BST07] and to a more relaxed notion of common subsumers called good common subsumers (gcs), which are still common subsumers, but not necessary least ones.

Before we discuss the technical details of the computation of common subsumers w.r.t. background terminologies, we take a look at how this inferences are employed for the tasks for ontology design and maintenance.

**Bottom-up Construction** of knowledge bases is one of the main motivation for investigating the lcs inference. Here the ontology engineer wants to design the ontology bottom-up, i.e., by proceeding from the most specific concepts to the most general ones. This should be supported by automatically generating concepts from descriptions of typical instances of the new concept (and, additionally, of intended subclasses in the hierarchy–if any).

**Generating Concept Descriptions** as a first draft for the concept the ontology engineer wants to add to the ontology, but finds it difficult to describe. Based on the desired position of this concept in the subsumption hierarchy, one can offer the lcs concept for the (prospective) direct subsumees as a candidate description.

**Structuring the Ontology** to improve the structure of an ontology by inserting intermediate concepts into the subsumption hierarchy. The ontology engineer can obtain support on how to describe such concepts by, again, computing the lcs concept for the (prospective) direct subsumees as a candidate description.

**Ontology Customization** by adapting an existing ontology to purposes of a specific application by making simple modifications. Since the ontology user is not an expert in ontology languages, he works with a simpler language than the one used to formulate the ontology and/or with graphical frame-like interfaces. This case is the prototypical application for computing a lcs w.r.t. a background terminology.

The TONES "common framework for representing ontologies"[CGG+06] distinguishes three kinds of ontologies. The task of computing common subsumers w.r.t. a background ontology refers rather to stand-alone ontologies than to peer ontologies. Although we distinguish two ontologies in use–the background and the user ontology–, this setting is not a typical setting for peer ontologies. The mapping between the ontologies is trivial in our case: it's identity. Furthermore, once the user ontology has been build, simply the union of the two ontologies is used.

## 11.2 Preliminaries

In order to devise the techniques for computing lcs or gcs w.r.t. background terminologies we need to refine some notions already introduced earlier.

From the DL $\mathcal{ALC}$which was introduced in Section 2 the DL $\mathcal{ALE}$is obtained by disallowing disjunction and restricting general negation to primitive negation, i.e. negation

may only appear in front of concept names. The DL $\mathcal{EL}$ in turn is obtained by restricting $\mathcal{ALE}$ further. This DL only allows for conjunction and existential restrictions.

In contrast to general concept inclusion $C \sqsubseteq D$, where $C$ can be an arbitrary concept, a *concept definition* $A \equiv C$ assigns a concept name $A$ to a complex concept $C$. A finite set of such definitions is called an *acyclic TBox* iff it is acyclic (i.e., no definition refers, directly or indirectly, to the name it defines) and unambiguous (i.e., each name has at most one definition). If the TBox is unambiguous, but not acyclic, then it is called a *cyclic TBox*. The concept names occurring on the left-hand side of a concept definition are called *defined* concepts, and the others *primitive*. In this section we call a TBox containing GCIs a *general TBox*. If we say just TBox then this means an acyclic, a cyclic or a general TBox. An acyclic or a cyclic $\mathcal{ALE}$-TBox must satisfy the additional restriction that no defined concept occurs negated in it (i.e., negation can only be applied to primitive concepts).

The interpretation $\mathcal{I}$ is a model of the (a)cyclic TBox $\mathcal{T}$ iff it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \equiv C$ in $\mathcal{T}$. It is a model of the general TBox $\mathcal{T}$ iff it satisfies all its concept inclusions, i.e., $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$ holds for all $C \sqsubseteq D$ in $\mathcal{T}$.

Given this semantics, we can now turn to the inferences. The subsumption relation $\sqsubseteq_{\mathcal{T}}$ is a preorder (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 preorder $\sqsubseteq_{\mathcal{T}}$ induces a partial order $\sqsubseteq_{\mathcal{T}}^{\equiv}$ on the equivalence classes of concepts:

$$[C_1]_{\equiv} \sqsubseteq_{\mathcal{T}}^{\equiv} [C_2]_{\equiv} \text{ iff } C_1 \sqsubseteq_{\mathcal{T}} C_2,$$

where $[C_i]_{\equiv} := \{D \mid C_i \equiv_{\mathcal{T}} D\}$ is the equivalence class of $C_i$ $(i = 1, 2)$.When talking about the *subsumption hierarchy*, we mean this induced partial order.

In addition to standard inferences like computing the subsumption hierarchy, so-called *non-standard inferences* have been introduced and investigated in the DL community (see, e.g., [Küs01]). In this section, we concentrate on the problem of computing the least common subsumer. Originally, this problem was introduced for concept descriptions (i.e., w.r.t. the empty TBox). In the presence of acyclic TBoxes, one can apply this inference if one first expands the concept descriptions. Let $\mathcal{L}$ be some description logic.

**Definition 11.1** [least common subsumer (lcs)] 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 $i = 1, \ldots, n$ *(D is a common subsumer);*
2. if $E$ is an $\mathcal{L}$-concept description satisfying $C_i \sqsubseteq E$ for $i = 1, \ldots, n$, then $D \sqsubseteq E$ *(D is least).*

As an easy consequence of this definition, the 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). Thus, it is enough to devise algorithms for computing the binary lcs. It should be noted, however, that the lcs need not always exist.

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 for our purpose. Instead of extracting properties common to $C_1, \ldots, C_n$, it just gives their disjunction, which does not provide the ontology engineer with new information. For the DLs introduced above, this means that 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 grow exponentially in the size of the input concept descriptions.

Let us now define the *new non-standard inference* introduced recently in [BST04b, BST04a], which is a generalization of the lcs to (a)cyclic or general background TBoxes. Let $\mathcal{L}_1, \mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-DL of $\mathcal{L}_2$, i.e., $\mathcal{L}_1$ allows for less constructors. For a given $\mathcal{L}_2$-TBox $\mathcal{T}$, we call $\mathcal{L}_1(\mathcal{T})$-*concepts* those $\mathcal{L}_1$-concepts that may contain concepts defined in $\mathcal{T}$.

**Definition 11.2** [least common subsumer w.r.t. background terminology] 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).*

The obtained concept description only uses concept constructors from $\mathcal{L}_1$, but it uses concept names defined in the $\mathcal{L}_2$-TBox. This is the main distinguishing feature of a least common subsumer w.r.t. a background terminology. 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. Let us illustrate this inference by a trivial example.

**Example 11.3** 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 $\top$.

It may seem 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 computing the approximation of $\mathcal{ALC}$ by $\mathcal{ALE}$(see [BKT02b]). To make this more precise, we define the non-standard inference of approximating concepts of one DL $\mathcal{L}_1$ by descriptions of a less expressive DL $\mathcal{L}_2$.

**Definition 11.4** [concept approximation] Given an $\mathcal{L}_2$-concept $C$, the $\mathcal{L}_1$-concept $D$ *approximates $C$ from above* iff $D$ is the least $\mathcal{L}_1$-concept satisfying $C \sqsubseteq D$.

In [BKT02b] 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 $\mathcal{ALE}(\mathcal{T})$-concepts $C_1, \ldots, C_n$, one can first expand $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$ to concepts $C'_1, \ldots, C'_n$. These are $\mathcal{ALC}$-concepts, 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 11.3 above, where the approach based on approximation that we have just sketched yields $\top$ rather than the lcs $A$.

## 11.3 Existence and non-existence of the lcs w.r.t. TBoxes

We start with the DL $\mathcal{EL}$as the formalism for the user ontology and examine the cases (i) of acyclic and unambiguous and of (ii) cyclic $\mathcal{ALC}$-ontologies. Then we extend the results to the user DL $\mathcal{ALE}$.

### 11.3.1 Existence and non-existence of the $\mathcal{EL}(\mathcal{T})$-lcs

In this section, we assume that $\mathcal{L}_1$ is $\mathcal{EL}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. In addition, we assume that the sets of concept and role names available for building concept descriptions are finite. First, we consider the case of acyclic TBoxes.

**Theorem 11.5** *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox. The lcs of $\mathcal{EL}(\mathcal{T})$-concept descriptions w.r.t. $\mathcal{T}$ always exists and can effectively be computed.*

The theorem is shown in [BST04a] and an easy consequence of the following facts:

1. If $D$ is an $\mathcal{EL}(\mathcal{T})$-concept description of role depth $k$, then there are (not necessarily distinct) roles $r_1, \ldots, r_k$ such that $D \sqsubseteq \exists r_1.\exists r_2. \ldots \exists r_k.\top$

2. Let $C$ be an $\mathcal{EL}(\mathcal{T})$-concept description, and assume that the $\mathcal{ALC}$-concept description $C'$ obtained by unfolding $C$ w.r.t. $\mathcal{T}$ is satisfiable and has the role depth $\ell < k$. Then $C' \not\sqsubseteq \exists r_1.\exists r_2. \ldots \exists r_k.\top$, and thus $C \not\sqsubseteq_{\mathcal{T}} \exists r_1.\exists r_2. \ldots \exists r_k.\top$. In fact, the standard tableau-based algorithm for $\mathcal{ALC}$ applied to $C'$ constructs a tree-shaped interpretation of depth at most $\ell$ whose root individual belongs to $C'$, but not to $\exists r_1.\exists r_2. \ldots \exists r_k.\top$.

3. For a given bound $k$ on the role depth, there is only a finite number of inequivalent $\mathcal{EL}$-concept descriptions of role depth at most $k$. This is a consequence of the fact that we have assumed that the sets of concept and role names are finite, and can be shown by induction on $k$.

It is not hard to see that the above facts point to a method how to one can effectively compute $\mathcal{EL}(\mathcal{T})$-lcs. First one enumerates all (representatives of the equivalence classes of) all common subsumers of $C_1, \ldots, C_n$, and then build their conjunction. However, this brute-force algorithm is probably not useful in practice.

Second, we consider the case of TBoxes allowing for GCIs.

**Theorem 11.6** *Let $\mathcal{T} := \{A \sqsubseteq \exists r.A,\ B \sqsubseteq \exists r.B\}$. Then, the lcs of the $\mathcal{EL}(\mathcal{T})$-concept descriptions $A, B$ w.r.t. $\mathcal{T}$ does not exist.*

So, already for the small DL $EL$ the lcs w.r.t. cyclic background knowledge bases does not exist.

### 11.3.2 Existence and non-existence of the $\mathcal{ALE}(\mathcal{T})$-lcs

In this section, we assume that $\mathcal{L}_1$ is $\mathcal{ALE}$and $\mathcal{L}_2$ is $\mathcal{ALC}$. The following Theorem is shown in [BST07].

**Theorem 11.7** *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.*

A first attempt to show Theorem 11.7 could be the following. Let $C_1, C_2$ be $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and assume that the role depths of the $\mathcal{ALC}$-concept description $C_1', C_2'$ obtained by expanding the descriptions $C_i$ w.r.t. $\mathcal{T}$ are bounded by $k$. If we could show as in the case of $\mathcal{EL}(\mathcal{T})$ that this implies that the role depth of any common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$ is also bounded by $k$, then we could obtain the least common subsumer by simply building the (up to equivalence) finite conjunction of all common subsumers of $C_1, C_2$ in $\mathcal{ALE}(\mathcal{T})$.

However, due to the fact that in $\mathcal{ALC}$ and $\mathcal{ALE}$we can define unsatisfiable concepts, this simple approach does not work. In fact, $\bot$ has role depth 0, but it is subsumed by any concept description. Given this counterexample, the next conjecture could be that it is enough to prevent this pathological case, i.e., assume that at least one of the concept descriptions $C_1, C_2$ is satisfiable w.r.t. $\mathcal{T}$, i.e., not subsumed by $\bot$ w.r.t. $\mathcal{T}$. This assumption can be made without loss of generality. In fact, if $C_1$ is unsatisfiable w.r.t. $\mathcal{T}$ (i.e., equivalent to $\bot$ w.r.t. $T$), then $C_2$ is the lcs of $C_1, C_2$ w.r.t. $T$.

In fact, to compute the lcs of $C_1, C_2$ w.r.t. $\mathcal{T}$, it is enough to compute the (up to equivalence) finite set of all $\mathcal{ALE}(\mathcal{T})$-concept descriptions of role depth at most $k+1$, check which of them are common subsumers of $C_1, C_2$ w.r.t. $\mathcal{T}$, and then build the conjunction $E$ of these common subsumers which is finite. By definition, $E$ is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$, and for any common subsumer $D$ of $C_1, C_2$ w.r.t. $\mathcal{T}$, there is a conjunct $D_0$ in $E$ such that $D_0 \sqsubseteq_{\mathcal{T}} D$, and thus $E \sqsubseteq_{\mathcal{T}} D$. Again, this brute-force algorithm is probably not useful in practice.

If we allow for general TBoxes $\mathcal{T}$, then the lcs w.r.t. $\mathcal{T}$ need not exist.

**Theorem 11.8** *Let $\mathcal{T} := \{A \sqsubseteq \exists r.A, B \sqsubseteq \exists r.B\}$, where $A, B$ are distinct concept names. Then, the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions $A, B$ w.r.t. $\mathcal{T}$ does not exist.*

It is easy to see that the same claim holds even for the cyclic TBox $\mathcal{T} := \{A \equiv \exists r.A, B \equiv \exists r.B\}$.

**Corollary 11.9** *Let $\mathcal{T} := \{A \equiv \exists r.A, B \equiv \exists r.B\}$, where $A, B$ are distinct concept names. Then, the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions $A, B$ w.r.t. $\mathcal{T}$ does not exist.*

Thus the lcs can in general not be extended to general TBoxes that contain GCIs. However, to support the ontology design and maintenance tasks mentioned at the beginning of this section it is not necessary to compute the *least* common subsumer, a good common subsumer may also suffice to solve the task.

## 11.4 Good common subsumers

The brute-force algorithm for computing the lcs in $\mathcal{ALE}(\mathcal{T})$w.r.t. an acyclic background $\mathcal{ALC}$-TBox described in the previous section is not useful in practice since the number of concepts 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 it may even result in over-fitting! A common subsumer that is not too general can also be used. In this section, 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.

### 11.4.1 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 $\mathcal{ALE}$-concepts $C, D$ (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:

$$\begin{array}{rcl@{\qquad}rcl}
\forall r.E \sqcap \forall r.F & \longrightarrow & \forall r.(E \sqcap F), & \forall r.E \sqcap \exists r.F & \longrightarrow & \forall r.E \sqcap \exists r.(E \sqcap F), \\
\forall r.\top & \longrightarrow & \top, & E \sqcap \top & \longrightarrow & E, \\
\exists r.\bot & \longrightarrow & \bot, & E \sqcap \bot & \longrightarrow & \bot, \\
A \sqcap \neg A & \longrightarrow & \bot \text{ for each } A \in \mathsf{N_C}. & & &
\end{array}$$

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 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 $\mathsf{restrict}_r^{\exists}(C)$ ($\mathsf{restrict}_r^{\forall}(C)$) 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

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

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.

### 11.4.2 A good common subsumer in $\mathcal{ALE}$w.r.t. a background TBox

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.[9] Let $C, D$ be normalized $\mathcal{ALE}(\mathcal{T})$-concept descriptions. We apply the above algorithm for $\mathcal{ALE}$-concept descriptions, but we 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

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

to obtain concept names in the result concept. 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 11.10** *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 $\mathcal{T}$, i.e. applying the original $\mathcal{ALE}$ lcs algorithm. though it need not be the least common subsumer. In the following, we will call the common subsumer computed this way *good common subsumer (gcs)*, and the algorithm that computes it the *gcs algorithm*.

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 quite 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 Section 11.5 we will show that attribute exploration can be used for this purpose.

### 11.4.3 Using $\mathcal{ALE}$-expansion when computing the gcs

If the background terminology is an acyclic TBox $\mathcal{T}$, then one can employ an appropriate partial expansion of $\mathcal{T}$ in order to uncover $\mathcal{ALE}$-parts hidden within the defined concepts. The idea is that the gcs algorithm will possibly yield a more specific common subsumer if it can make use of $\mathcal{ALE}$-concepts "hidden" within the defined concepts.

However,as a simple example, consider the $\mathcal{ALC}$-TBox $\mathcal{T}$:

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

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

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

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

For instance, the concepts defining NoSon and NoDaughter are actually $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and thus $C, D$ can be expanded to

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

before computing the gcs. The concepts defining DaughterHappyDoctor and SonRichDoctor are not $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and thus these two names cannot be expanded. However, in this example, the common subsumer computed by applying the gcs algorithm to the expanded concepts $C', D'$ is

$$\exists \mathsf{has\text{-}child}.\top,$$

which is actually less specific than $\exists$has-child.ChildrenDoctor the result of applying the gcs algorithm to the unexpanded concepts $C, D$.

To overcome this problem, we do the partial expansion, but also keep the defined concepts that we have expanded. In the example, this yields the expanded concepts

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

If we apply the gcs algorithm to $C'', D''$, then we obtain (up to equivalence w.r.t. $\mathcal{T}$) the same common subsumer as obtained from $C, D$, i.e., in this case the expansion does not yield a more specific result.

However, it is easy to construct examples where this kind of expansion leads to better results. For instance, if we apply the gcs algorithm to $\forall$has-child.(Female $\sqcap$ Doctor) and NoSon $\sqcap \forall$has-child.Happy, then the result is $\top$. In contrast, if we apply it to the expanded concept descriptions $\forall$has-child.(Female $\sqcap$ Doctor) and NoSon $\sqcap \forall$has-child.Female $\sqcap$ $\forall$has-child.Happy, then the result is the more specific common subsumer $\forall$has-child.Female.

Before checking whether a defined concept can be expanded, it is useful to transform it into *negation normal form (NNF)* by pushing all negations into the description until they occur only in front of concept names, using de Morgan' rules and the facts that $\neg\neg D \equiv D$ and $\neg\top \equiv \bot$. For example, the concept description $\neg\forall$has-child.Female is not an $\mathcal{ALE}$-concept description, but its negation normal form $\exists$has-child.$\neg$Female is. More formally, we define the $\mathcal{ALE}$*-expansion* of (negated) concept names defined in $\mathcal{T}$ and of $\mathcal{ALE}(\mathcal{T})$-concept descriptions as follows.

**Definition 11.11** [$\mathcal{ALE}$-expansion] Let $\mathcal{T}$ be an acyclic TBox, let $A$ be a concept name defined in $\mathcal{T}$, and let $A \equiv C$ be its definition. We first build the negation normal form $C'$ of $C$. If $C'$ is not an $\mathcal{ALE}(\mathcal{T})$-concept description, then the $\mathcal{ALE}$-expansion of $A$ is $A$. Otherwise, it is $A \sqcap C''$, where $C''$ is obtained from $C'$ by replacing all (negated) defined concept names in $C'$ by their $\mathcal{ALE}$-expansion.

To obtain the $\mathcal{ALE}$-expansion of $\neg A$, we just apply the same approach to $\neg C$. The $\mathcal{ALE}$-expansion of an $\mathcal{ALE}(\mathcal{T})$-concept description is obtained by replacing all (negated) defined concept names by their $\mathcal{ALE}$-expansions.

Note that this recursive definition of an $\mathcal{ALE}$-expansion is well-founded since the TBox is assumed to be acyclic. As an example, consider the TBox $\mathcal{T}$ consisting of

$$A \equiv \neg\forall r.(B_1 \sqcup B_2), \quad B_1 \equiv P \sqcup Q, \quad B_2 \equiv P \sqcap Q.$$

Then we obtain $A \sqcap \exists r.(\neg B_1 \sqcap \neg P \sqcap \neg Q \sqcap \neg B_2)$ as $\mathcal{ALE}$-expansion of $A$.

It is easy to see that $\mathcal{ALE}$-expansion may lead to more specific common subsumers, but never to less specific ones.

**Proposition 11.12** *Let $\mathcal{T}$ be an acyclic $\mathcal{L}_2$-TBox, $C, D$ $\mathcal{ALE}(\mathcal{T})$-concept descriptions with $\mathcal{ALE}$-expansion $C', D'$, and let $E$ ($E'$) be the result of applying the gcs algorithm to $C, D$ ($C', D'$). Then $E'$ is a common subsumer of $C, D$ that is at least as good as $E$, i.e., $C \sqsubseteq_{\mathcal{T}} E'$, $D \sqsubseteq_{\mathcal{T}} E'$, and $E' \sqsubseteq_{\mathcal{T}} E$.*

$\mathcal{ALE}$-expansion can also be applied to cyclic $\mathcal{L}_2$-TBoxes, provided that the cycles go through non-$\mathcal{ALE}$parts of the TBox. This is, for example, the case in the TBox $\mathcal{T} := \{A \equiv \exists r.B,\ B \equiv P \sqcup A\}$.

### 11.4.4 Alternative approaches for computing common subsumers

In Section 11.2 we have already sketched an approach based on approximation, which works if the TBox $\mathcal{T}$ is acyclic, $\mathcal{L}_2$ allows for disjunction, and one can compute the approximation from above of $\mathcal{L}_2$-concept descriptions by $\mathcal{ALE}$-concept descriptions. For example, if we take $\mathcal{ALC}$ as $\mathcal{L}_2$, then all these conditions are satisfied.

**Definition 11.13** [Common subsumer by approximation (acs)] Assume that $\mathcal{L}_2$ allows for disjunction, and that one can compute the approximation from above of $\mathcal{L}_2$-concept descriptions by $\mathcal{ALE}$-concept descriptions. Let $\mathcal{T}$ be an acyclic $\mathcal{L}_2$-TBox. Given $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C, D$, one first fully expands them into $\mathcal{L}_2$-concept descriptions $C', D'$. Then one approximates their disjunction $C' \sqcup D'$ from above by an $\mathcal{ALE}$-concept description. The common subsumer of $C, D$ obtained this way is called the *common subsumer by approximation (acs)* of $C, D$ w.r.t. $\mathcal{T}$.

In Section 11.2, we have shown by an example that the acs can be less specific than the lcs. In this example (Example 11.3), the gcs coincides with the lcs, and thus is also more specific than the acs: in fact, w.r.t. the TBox $\mathcal{T} = \{A \equiv P \sqcup Q\}$, the smallest conjunction of concept names above both $P$ and $Q$ is $A$, and thus the gcs of $P$ and $Q$ is $A$.

There are, however, also examples where the gcs is less specific than the acs. For instance, consider the TBox

$$\mathcal{T} = \{A \equiv \exists r.A_1 \sqcup \exists r.A_2, \quad B \equiv \exists r.B_1 \sqcup \exists r.B_2\}.$$

With respect to this TBox, the gcs of $A, B$ is $\top$, whereas the acs is the more specific common subsumer $\exists r.\top$.

The gcs algorithm makes use of the subsumption relationships between conjunctions of (negated) concept names. Usually, these relationships are not known for a given TBox,

and thus we must either precompute them (see Section 11.5) or compute them on the fly (as sketched in Section 11.4.2). Both may be quite expensive. What is usually known for a given TBox $\mathcal{T}$ are all subsumption relationships between the concept names occurring in $\mathcal{T}$.[10] This information can be used as follows. Given two conjunctions

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

the gcs algorithm takes 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 conjunctions. In contrast, the algorithm that just ignores the TBox would take

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

Using the subsumption relationships between concept names, we can come up with a new approach that lies between these two approaches.

**Definition 11.14** [Subsumption closure] Let $\mathcal{T}$ be a TBox, and $S$ ($\overline{S}$) a set of (negated) concept names. The *subsumption closure* of $S$ ($\overline{S}$) w.r.t. $\mathcal{T}$ is a set of (negated) concept names, which is defined as follows:

$$\begin{array}{lcl} \mathsf{SC}(S) & := & \{A \mid \exists B \in S.\ B \sqsubseteq_{\mathcal{T}} A\}, \\ \mathsf{SC}(\overline{S}) & := & \{\neg A \mid \exists \neg B \in \overline{S}.\ A \sqsubseteq_{\mathcal{T}} B\}. \end{array}$$

Instead of using the intersection $\mathsf{names}(C) \cap \mathsf{names}(D)$ ($\overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)$), as in the approach that ignores $\mathcal{T}$, one can first build the subsumption closures, and then intersect the closures, i.e., use

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

We call the algorithm for computing common subsumers obtained this way the *scs algorithm*, and the result of applying it to $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C, D$ the *scs of* $C, D$ *w.r.t.* $\mathcal{T}$.

**Proposition 11.15** *Let $\mathcal{T}$ be an $\mathcal{L}_2$-TBox, $C, D$ $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and let $E$ ($E'$) be the result of applying the gcs (scs) algorithm to $C, D$. Then $E'$ is a common subsumer of $C, D$ that is at most as good as the gcs $E$, i.e., $C \sqsubseteq_{\mathcal{T}} E'$, $D \sqsubseteq_{\mathcal{T}} E'$, and $E \sqsubseteq_{\mathcal{T}} E'$.*

To obtain a practical gcs algorithm, we must be able to compute the smallest conjunction of (negated) concept names that subsumes two such conjunctions w.r.t. $\mathcal{T}$ in an efficient way. Since in our application scenario (customization of DL knowledge bases based on 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 a realistic size. Instead, we propose to use attribute exploration for this purpose.

[10] Most DL system compute these automatically when reading in a TBox.

## 11.5 Computing the subsumption lattice of conjunctions of (negated) concept names w.r.t. a TBox

In the following we refer to notions from formal concept analysis that were already introduced in Section 10.2 of this deliverable.

In order to apply attribute exploration (this method was discussed in Section 10.2, Algorithm 1) to the task of computing the subsumption lattice[11] 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 (for a general definition see Definition 10.1) whose concept lattice is isomorphic to the subsumption lattice we are interested in. 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 11.16** Let $\mathcal{T}$ be an $\mathcal{L}_2$-TBox. The context $\mathcal{K}_\mathcal{T} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:

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

The following Theorem was proven in [BST07].

**Theorem 11.17** *The concept lattice of the context $\mathcal{K}_\mathcal{T}$ is isomorphic to the subsumption hierarchy of all conjunctions of subsets of $\mathcal{P}$ w.r.t. $\mathcal{T}$.*

Attribute exploration can be used to compute the concept lattice of $\mathcal{K}_\mathcal{T}$ since any standard subsumption algorithm for the DL under consideration is an expert for $\mathcal{K}_\mathcal{T}$. In fact, any decision procedure for subsumption w.r.t. TBoxes in $\mathcal{L}_2$ functions as an expert for the context $\mathcal{K}_\mathcal{T}$.

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 $\overline{A}_1 \equiv \neg A_1, \ldots, \overline{A}_n \equiv \neg A_n$.[12] The concept lattice of the context $\mathcal{K}_{\widehat{\mathcal{T}}}$ is isomorphic to the subsumption hierarchy of all conjunctions of concept names and negated concept names occurring in $\mathcal{T}$.

Thus Attribute exploration applied to $\mathcal{K}_{\widehat{\mathcal{T}}}$ can be used to compute the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$. Given this base, we can compute the supremum in the concept lattice of $\mathcal{K}_{\widehat{\mathcal{T}}}$ as follows:

[11] 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.

[12] For $\widehat{\mathcal{T}}$ to be an $\mathcal{L}_2$-TBox, we must assume that $\mathcal{L}_2$ allows for full negation.

**Lemma 11.18** *Let $\mathcal{J}$ be the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$, and let $B_1, B_2$ be sets of attributes of $\mathcal{K}_{\widehat{\mathcal{T}}}$. Then $\mathcal{J}(B_1) \cap \mathcal{J}(B_2)$ is the intent of the supremum of the formal concepts $(B_1', B_1'')$ and $(B_2', B_2'')$.*

As an immediate consequence of this lemma together with Theorem 11.17 and its proof, 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 11.19** *Let $\mathcal{J}$ be the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$, 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 $\bigsqcap_{L \in B_2} L$.*

Computing the implication hull $\mathcal{J}(B)$ for a set of attributes $B$ can be done in time linear in the size of $\mathcal{J}$ and $B$. This means that the supremum can be computed efficiently as long as the Duquenne-Guigues base of $\mathcal{K}_{\widehat{\mathcal{T}}}$ is relatively small.

## 11.6 Future and Related Work

The problem of computing the lcs has already been investigated in the literature for a couple of Description Logics. One can divide the algorithms for computing the lcs according the kind of terminology the algorithms support. For acyclic and unambiguous TBoxes the lcs has been investigated in [CH94, FP96, BKM99, KM01, KB01, BTK03]. Algorithms for computing the lcs w.r.t. cyclic TBoxes and w.r.t. greatest fixed point semantics were proposed in [BK98, Baa03b]. For TBoxes that use GCIs the lcs has so far not been investigated. The results presented here indicate that for $\mathcal{EL}$ or $\mathcal{ALE}$ the lcs need not exist. However, most applications use TBoxes that contain GCIs. To support ontology maintenance tasks, it is desirable to compute commonalities in the presence of GCIs. The here presented approach for computing a gcs, more precisely the scs, can bridge this gap to some extend. The computation algorithm for (unfolded) concept descriptions devised here can also be applied to background terminologies that contain GCIs, since the method only makes use of subsumption w.r.t. the background terminology. However, the task of how to "unfold" the input concepts, i.e., to make the information captured in the terminology explicit, remains to be solved.

Furthermore, on the theoretical side, the main topic for future research is to find exact algorithms for computing the *least* common subsumer that are better than the brute-force algorithm sketched in the proof of Theorem 11.7.

On the practical side, we integrate an implementation of the scs computation into our non-standard inference system sonic [TK04]. The integration of the scs in sonic will enable us to see whether this fairly inexpensive way of computing a common subsumer is already useful for practical applications, or whether the more expensive gcs or lcs is needed.

# 12 Ontology Extraction from DB Schemas

## 12.1 Introduction

A number of important database problems have been shown to have improved solutions by using a conceptual model or an ontology to provide *precise semantics* for a database schema. These scenarios include federated databases, data warehousing [CDGL$^+$01], information integration through mediated schemas [Len02], and the Semantic Web [HH01] (for a survey see [WVV$^+$01]). Since ontologies provide a conceptual view of the application domain, the recent trend to employ such ontologies for navigational (and reasoning) purposes when accessing the data gives additional motivation for the problem of extracting the ontology from database schema [LLS06]. When such an ontology exists, modeling the relation between the data sources and an ontology is a crucial aspect in order to capture the semantics of the data.

In this chapter we define the framework for extracting from a relational database an ontology that is to be used as a conceptual view over the data, where the semantic mapping between the database schema and the ontology is captured by associating a view over the source data to each element of the ontology. Thus, the vocabulary over the data can be seen as a set of (materialized) views over the vocabulary of the ontology [FTG$^+$06] (i.e., technique known as GAV approach in the literature [Len02]).

The heuristics that underlie the ontology extraction process are based on a careful study of standard design process relating the constructs of the relational model with those of conceptual modeling languages. In particular, we use ideas of standard relational schema design from Entity-Relationship diagrams.

Our proposed ontology extraction algorithm works in two phases. First, the classification schemes for tables necessary for the extraction process are derived. Second, we devise a process for extracting the an ontology describing the data source; plus the set of view definitions, which express the mapping between the database schema and the ontology.

The rest of the chapter is structured as follows. In section 2 we present the formal framework, where first the constraints expressed over the relational schema are defined, and the ontology language is presented. Section 12.3 describes an intuitive progression of ideas underlying our approach, while section 12.4 provides the ontology extraction algorithm. In section 12.5 we shortly report on related work.

## 12.2 Preliminaries

We introduce the formal framework for representing ontologies and their relation with a relational data source. We call a *DLR-DB* system $\mathcal{S}$ is a triple $\langle \mathcal{R}, \mathcal{P}, \mathcal{K} \rangle$, where $\mathcal{R}$ is a *relational schema*, $\mathcal{P}$ is a *component structure* over $\mathcal{R}$, and $\mathcal{K}$ is a set of assertions involving names in $\mathcal{R}$. In this section we describe these concepts.

### 12.2.1 Relational Model

We make use of the standard notion of relational model by using named attributes, each with an associated datatype, instead of tuples.

**Definition 12.1** *A* relational schema $\mathcal{R}$ *is a set of relationships, each one with a fixed set of attributes (assumed to be pairwise distinct) with associated datatypes. We use* $[s_1 : D_1, \ldots, s_n : D_n]$ *to denote that a relationship has attributes* $s_1, \ldots, s_n$ *with associated data types* $D_1, \ldots, D_n$*. We interpret relationships over a fixed countable* domain $\Delta$ *of datatype elements, which we consider partitioned into the datatypes* $D_i$*. The domain contains a special constant (*$\mathsf{null}$*) called the* NULL value*.*

*A* database instance *(or simply* database*)* $\mathcal{D}$ *over a relational schema* $\mathcal{R}$ *is an (interpretation) function that maps each relationship* $R$ *in* $\mathcal{R}$ *into a set* $R^{\mathcal{D}}$ *of total functions from the set of attributes of* $R$ *to* $\Delta$*. The instance must satisfy the attribute datatypes specified in the relational schema. I.e., if* $R$ *has attributes* $[s_1 : D_1, \ldots, s_n : D_n]$*, and* $\phi \in R^{\mathcal{D}}$*, then* $\phi(s_i) \in D_i$ *or* $\phi(s_i) = \mathsf{null}$*.*

To ease the presentation of the semantics for a *DLR-DB* system we make use of relational algebra. To this purpose, we introduce the definition of a projection of a relationship over a sequence of attributes.

**Definition 12.2** *Let* $A = [s_1, \ldots, s_m]$ *be a sequence of* $m$ *attribute names of a relationship* $R$ *of a schema* $\mathcal{R}$*, and* $\mathcal{D}$ *a database over* $\mathcal{R}$*. The* projection *of* $R^{\mathcal{D}}$ *over* $A$ *is the relation* $\pi_A R^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^m$*, satisfying the condition that* $\phi \in R^{\mathcal{D}}$ *iff* $(\phi(s_1), \ldots, \phi(s_n)) \in \pi_A R^{\mathcal{D}}$*.*

Note that the order of the attributes in $A$ fixes the correspondence between positional arguments of $\pi_A R^{\mathcal{D}}$ and the attribute names of $R$.

### 12.2.2 Ontology Language

The ontology language enables the explicit representation of the constraints which the underlying data source satisfies. The use of an ontology language can be seen as an alternative to the use of standard modelling paradigms of Entity Relationships or UML diagrams. The advantage over these formalisms lies on the fact that the ontology language has a clear and unambiguous semantics which enables the use of automatic reasoning to support the designer.

In addition to the standard definition of a relational schema, we introduce the concept of named components. The intuition behind a named component is the role name of a relationship in an ER schema (or UML class-diagram). The *component structure* $\mathcal{P}$ associates to each relationship a mapping from *named components* to sequences of attributes. Let $R$ be a relationship in $\mathcal{R}$, to ease the notation we write $\mathcal{P}_R$ instead of $\mathcal{P}(R)$.

**Definition 12.3** *Let* $R$ *be a relationship in* $\mathcal{R}$*, with attributes* $[s_1 : D_1, \ldots, s_n : D_n]$*.* $\mathcal{P}_R$ *is a nonempty (partial) function from a set of named components to the set of nonempty sequences of attributes of* $R$*. The domain of* $\mathcal{P}_R$*, denoted* $\mathcal{C}_R$*, is called the set of* components of $R$*. For a named component* $c \in \mathcal{C}_R$*, the sequence* $\mathcal{P}_R(c) = [s_{i_1}, \ldots, s_{i_m}]$*, where each* $i_j \in \{1, \ldots, n\}$*, is called the* $c$-component *of* $R$*. We denote with* $\delta(R)$ *the set of all attributes not belonging to any* $c$*-component.*

*We require that the sequences of attributes for two different named components are not overlapping, and that each attribute appears at most once in each sequence. I.e., given* $\mathcal{P}_R(c_i) = [s_{i_1}, \ldots, s_{i_k}]$ *and* $\mathcal{P}_R(c_j) = [s_{j_1}, \ldots, s_{j_m}]$*, if* $s_{i_\ell} = s_{j_r}$ *then* $c_i = c_j$ *and* $\ell = r$*.*

*The* signature *of a component $\mathcal{P}_R(c)$, denoted $\tau(\mathcal{P}_R(c))$, is the sequence of types of the attributes of the component. Specifically, if the attributes of $R$ are $[s_1 : D_1, \ldots, s_n : D_n]$, the signature of the component $\mathcal{P}_R(c) = [s_{i_1}, \ldots, s_{i_m}]$ is the sequence $[D_{i_1}, \ldots, D_{i_m}]$.*

*Two components $\mathcal{P}_R(c_1)$ and $\mathcal{P}_R(c_2)$ are* compatible *if the two signatures $\tau(\mathcal{P}_R(c_1))$ and $\tau(\mathcal{P}_R(c_2))$ are equal.*

The *DLR-DB* ontology language, used to express the constraints in $\mathcal{K}$, is based on the idea of modelling the domain by means of *axioms* involving the projection of the relationship over the named components. The constraints are intended to capture typical constructs encountered in conceptual models, in fact it could be considered a sort of textual representation of ER diagrams.

An *atomic formula* is a projection of a relationship $R$ over one of its components. The projection of $R$ over the $c$-component is denoted by $R[c]$. When the relationship has a single component, then this can be omitted and the atomic formula $R$ corresponds to its projection over the single component.

Two atomic formulae $R[c]$ and $S[c']$ are *compatible* iff the two corresponding components $\mathcal{P}_R(c)$ and $\mathcal{P}_S(c')$ are compatible. Given the atomic formulae $R[c], R'[c'], R_i[c_i]$, an *axiom* is an assertion of the form

$$\begin{array}{rl} R[c] \sqsubseteq R'[c'] & \text{Subclass} \\ R[c] \text{ disj } R'[c'] & \text{Disjointness} \\ \text{funct}(R[c]) & \text{Functionality} \\ R_1[c_1], \ldots, R_k[c_k] \text{ cover } R[c] & \text{Covering} \end{array}$$

where all the atomic formulae involved in the same axiom must be compatible.

The semantics of a *DLR-DB* system $\langle \mathcal{R}, \mathcal{P}, \mathcal{K} \rangle$ is given in terms of relational models for $\mathcal{R}$, where $\mathcal{K}$ plays the role of constraining the set of "admissible" models.

**Definition 12.4** *A database $\mathcal{D}$ for a schema $\mathcal{R}$ satisfies the axiom*

$$\begin{array}{rll} R[c] \sqsubseteq R'[c'] & \textit{if} & \pi_c R^{\mathcal{D}} \subseteq \pi_{c'} R'^{\mathcal{D}} \\ R[c] \text{ disj } R'[c'] & \textit{if} & \pi_c R^{\mathcal{D}} \cap \pi_{c'} R'^{\mathcal{D}} = \emptyset \\ \text{funct}(R[c]) & \textit{if} & \textit{for all } \phi_1, \phi_2 \in R^{\mathcal{D}} \textit{ with } \phi_1 \neq \phi_2, \\ & & \textit{we have } \phi_1(s) \neq \phi_2(s) \textit{ for some } s \textit{ in } c \\ R_1[c_1], \ldots, R_k[c_k] \text{ cover } R[c] & \textit{if} & \pi_c R^{\mathcal{D}} = \bigcup_{i=1\ldots n} \pi_{c_i} R_i'^{\mathcal{D}} \end{array}$$

*A database $\mathcal{D}$ is said to be a* model *for $\mathcal{K}$ if it satisfies all its axioms, and for each relations $R$ in $\mathcal{R}$ with components $c_1, \ldots, c_k$, for any $\phi_1, \phi_2 \in R^{\mathcal{D}}$ with $\phi_1 \neq \phi_2$, there is some $s$ in $c_i$ s.t. $\phi_1(s) \neq \phi_2(s)$.*

The above conditions are well defined because we assumed the compatibility of the atomic formulae involved in the constraints. Note that, in the definition above, we require the satisfiability of all the axioms, and in addition we consider the sequence of attributes of

all the components of a relationship as a key for the relationship itself. This reflects the fact that in conceptual models the additional attributes not belonging to any component are not considered relevant to identify an element of an entity or a relationship.

The *DLR-DB* ontology language enables the use of the most commonly used constructs in conceptual modelling. In particular, among these we mention:

**ISA,** using assertions of the form $E_1 \sqsubseteq E_2$, stating that the class $E_1$ is a subclass of the class $E_2$;[13]

**Disjointness,** using assertions of the form $E_1$ disj $E_2$, stating disjointness between the two classes $E_1$ and $E_2$;

**Role typing,** using assertions of the form $R[c] \sqsubseteq E$, stating that the role corresponding to the $c$ component of the relationship $R$ is of type $E$;

**Participation constraints,** using assertions of the form $E \sqsubseteq R[c]$, stating that instances of class $E$ participate to the relationship $R$ as value for the $c$ component;

**Non-participation constraints,** using assertions of the form $E$ disj $R[c]$, stating that instances of class $E$ do not participate to the relationship $R$ as value for the $c$ component;

**Functionality,** using assertions of the form funct$(R[c])$, stating that an object can appear in the $c$ component of the relationship $R$ at most once;

**Covering,** using the corresponding assertion to state that each member of a class must be contained in (at least) one of the covering classes.

Note that by taking away the covering axioms and considering only components containing single attributes this ontology language corresponds to *DLR-Lite* (see [CGL$^+$06]). The discussion on the actual reasoning tasks which can be employed in the context of *DLR-DB* systems is not in the scope of this document. Herewith we are mainly interested of the use of the language to express data models extracted from the relational data sources.

### 12.2.3 Relational Schemata

We assume that the reader is familiar with standard relational database notions as presented, for example, in [AHV95]. We assume that the database domain is a fixed denumerable set of elements $\Delta$ and that every such element is denoted uniquely by a constant symbol, called its *standard name* [LL01].

**Definition 12.5** *A relational schema $\mathcal{R}$ is a pair $(\Psi, \Sigma)$, where $\Psi$ is a set of relations, each with a fixed set of attributes (assumed to be pairwise disjoint) with associated datatypes. We use $[A_1 : D_1, \ldots, A_n : D_n]$ to denote that a relation has attributes*

[13] As mentioned in Section 12.2.2, when a relation has a single component, the component name can be omitted.

$A_1, \ldots, A_n$ *with associated datatypes* $D_1, \ldots, D_n$. $\Sigma$ *is a set of* integrity constraints *expressed on the relations in* $\Psi$*; i.e., assertions on the relations in* $\Psi$ *that express conditions that are intended to be satisfied by database instances.*

The semantics of relational schemata is provided in the usual way by means of the relational model, as described in Section 12.2.1. In our framework we consider the following kinds of integrity constraints:

- *nulls-not-allowed constraints*: given a relation $r$ in the schema, a nulls-not-allowed constraint over $r$ is an assertion of the form $nonnull(r, \mathbf{A})$, where $\mathbf{A}$ is a sequence of attributes of $r$. Such a constraint is satisfied in a database $\mathcal{D}$ if for each $\phi \in r^{\mathcal{D}}$ we have $\phi(a) \neq \mathsf{null}$ for each $a \in \mathbf{A}$.

- *unique constraints*: given a relation $r$ in the schema, a unique constraint over $r$ is an assertion of the form $unique(r, \mathbf{A})$, where $\mathbf{A}$ is a sequence of attributes of $r$. Such a constraint is satisfied in a database $\mathcal{D}$ if for each $\phi_1, \phi_2 \in r^{\mathcal{D}}$, with $\phi_1 \neq \phi_2$, we have $\phi_1(\mathbf{A}) \neq \phi_2(\mathbf{A})$.[14] When we have $unique(r, \mathbf{A})$ and $nonnull(r, \mathbf{A})$ for $r$, then a *key constraint* $key(r, \mathbf{A})$ is associated to $r$.

- *inclusion dependencies*: an inclusion dependency is an assertion of the form $r_1[\mathbf{A_1}] \subseteq r_2[\mathbf{A_2}]$, where $r_1, r_2$ are relations, $\mathbf{A_1}$, $\mathbf{A_2}$ are sequences of distinct attributes of $r_1$ and $r_2$, respectively. Such a constraint is satisfied in a database $\mathcal{D}$ if $\pi_{\mathbf{A_1}}(r_1^{\mathcal{D}}) \subseteq \pi_{\mathbf{A_2}}(r_2^{\mathcal{D}})$. We call an inclusion dependency $r_1[\mathbf{A_1}] \subseteq r_2[\mathbf{A_2}]$ where $\mathbf{A_2}$ is in $key(r_2, \mathbf{A_2})$ a *foreign key constraint*.

- *exclusion dependencies*: an exclusion dependency is an assertion of the form $(r_1[\mathbf{A_1}] \cap \ldots \cap r_m[\mathbf{A_m}]) = \emptyset$, where $m \geq 2$ $r_1, \ldots, r_m$ are relations, $\mathbf{A_1}, \ldots, \mathbf{A_m}$ are sequences of attributes of $r_1, \ldots, r_m$, respectively. Such a constraint is satisfied in a database $\mathcal{D}$ if $\pi_{\mathbf{A_1}}(r_1^{\mathcal{D}}) \cap \ldots \cap \pi_{\mathbf{A_m}}(r_m^{\mathcal{D}}) = \emptyset$.

- *covering constraints*: a covering constraint is an assertion of the form $(r_1[\mathbf{A_1}] \cup \ldots \cup r_m[\mathbf{A_m}]) = r_0[\mathbf{A_0}]$, where $m \geq 2$, $r_1, \ldots, r_m, r_0$ are relations, $\mathbf{A_1}, \ldots, \mathbf{A_m}, \mathbf{A_0}$ are sequences of attributes of $r_1, \ldots, r_m, r_0$, respectively. Such a constraint is satisfied in a database $\mathcal{D}$ if $\pi_{\mathbf{A_1}}(r_1^{\mathcal{D}}) \cup \ldots \cup \pi_{\mathbf{A_m}}(r_m^{\mathcal{D}}) = \pi_{\mathbf{A_0}}(r_0^{\mathcal{D}})$.

We denote sets of nulls-not-allowed, unique, key, inclusion and foreign key constraints expressed on a relation $r$ in $\Psi$ by $\mathcal{N}(r)$, $\mathcal{U}(r)$, $\mathcal{K}(r)$, $\mathcal{I}(r)$ and $\mathcal{F}(r)$, respectively. The set of exclusion and covering constraints expressed on relational schema are denoted by $\mathcal{E}$ and $\mathcal{C}$, respectively.

In the following we will use terms "table" and "column" when talking about relational schemas, reserving "relation(ship)" and "attribute" for aspects of ontologies. Thus, we will use the notation $T(\underline{K}, X)$ to represent a relational table $T$ with columns $KX$ and primary key $K$ (i.e., $key(T) = K$). Our notational convention is that single column names are indexed and appear in lower-case. A foreign key in $T$ is a set of columns $F$ that *references* the primary key $K'$ of table $T'$, and imposes a foreign key constraint $T[F] \subseteq T'[K']$.

[14] Given a function $\phi$ and a sequence of attributes $A = [s_1, \ldots, s_m]$, we use the notation $\phi(A)$ to indicate the tuple composed by the values of the attributes; i.e. $(\phi(s_1), \ldots, \phi(s_n))$.

## 12.3 Principles of conceptual schema extraction

Given a relational schema $\mathcal{R}$, our task is to extract from $\mathcal{R}$ the ontology in terms of *DLR-DB* ontology language, together with a set of views $\mathcal{V}$. Specifically, suppose we are given a table $T(\underline{k}, a, f)$ with key $k$, foreign key $f$ and non null values on the foreign key $f$. Then our goal is to identify for $T$ "reasonable" constructs in *DLR-DB* $\mathcal{K}$, taking also into account the constraints associated to $T$ (such as non null values for $f$).

We have chosen to flesh out the above principles in a systematic manner by considering the behavior of our proposed conceptual schema extraction algorithm on relational schemas designed from ER diagrams - a standard database modeling technique, widely covered in [BCN92, EN04], for example. One benefit of this approach is that it can be immediately shown that our algorithm, though heuristic in general, is correct for a certain class of schemas (e.g., when table schemes have not been denormalized). Also note that the assumption that a given relational schema was designed from some ER conceptual model does not mean that obtained ontology is this ER model, or is even expressed in the ER notation. In fact, we exploit the similarities of ER model and *DLR-DB* language over $\mathcal{K}$ for representing the knowledge about the domain.

We assume the reader is familiar with basics of ER modeling and database design [EN04, TLN06], though we summarize the ideas. We consider an ER model that supports entity sets $E$ (or simply entities) with attributes (referred to by $\mathsf{attrs}(E)$), $n$-ary ($n \geq 2$) relationship sets (or simply relationships) with attributes and ISA hierarchies between entities. In order to avoid ambiguities in the extraction process, we do not cover the case when sub-entities participate in a functional relationship. We refer to a sub-entity $S$ having $E$ as its super-entity by $S(E)$. Relationships are subject to cardinality constraints, which are used here in *min-max* notation [BCN92] and allow 1 as lower bounds (called *total* relationships), and 1 as upper bounds (called *functional* relationships). As a special case of $n$-ary relationships, we analyze in detail binary relationships as a reason for their wide use in ER diagrams, as well as UML class diagrams. A binary relationship $R$ between entities $E_1$ and $E_2$ (we denote such relationship by $R(E_1, E_2)$) is *one-to-one* (1:1), if it is functional on both directions (i.e., from $E_1$ to $E_2$ and from $E_2$ to $E_1$). $R(E_1, E_2)$ is *one-to-many* (1:N) if it is functional from $E_1$ to $E_2$, and $R(E_1, E_2)$ is *many-to-many* (N:M) if nether it nor its inverse is functional. A relationship $R$ from entity $E$ to itself is called *recursive* relationship, and is considered here in the form $R(E, E)$ (i.e., as a special case of binary relationship). Sub-entities of ISA may be constrained to be *disjoint* and/or *covering*. An entity $E$ has some attributes that act as identifier. We will refer to these using $\mathsf{id}(E)$ when describing the rules of schema design.

Table 7 specifies a mapping $\tau(C)$ (we follow mostly [BCN92, TLN06]) returning a relational table scheme being in *third normal form* (3NF)[15] for every ontology component $C$, where $C$ is either an atomic concept/entity or a $n$-ary relation(ship) ($n \geq 2$). Note that $\tau$ is defined recursively, and will only terminate if there are no cycles in a given conceptual model [MM90]. Also observe that one conceptual model may result in several

[15]The main reason for 3NF requirement is that it simplifies the ontology extraction process because each table will correspond to one atomic concept/entity and one or several binary relation(ships), rather than corresponding to several merged atomic concepts/entities. Anyhow, it should not be seen as a drawback, since 3NF modeling is a classical relational database modeling technique that minimizes data redundancy and, at the same time, is generally met without creating an exceedingly complicated schema.

| **ER component** $C$ | **Relational table** $\tau(C)$ |
|---|---|
| **Entity** $E$ | **Table** $\tau(E)$ |
| $X = \mathsf{attrs}(E)$ | columns: $X$ |
| $K = \mathsf{id}(E)$ | $key(\tau(E), K)$ |
| **Sub-entity** $S(E)$ | **Table** $\tau(S)$ |
| $X = \mathsf{attrs}(S)$ | columns: $KX$ |
| $K = \mathsf{id}(E)$ | $key(\tau(S), K)$ |
| | $\tau(S)[K] \subseteq \tau(E)[K]$ |
| if $S_1(E), \ldots, S_m(E)$ are disjoint | $(\tau(S_1)[K] \cap \ldots \cap \tau(S_m)[K]) = \emptyset$ |
| if $S_1(E), \ldots, S_m(E)$ are covering | $(\tau(S_1)[K] \cup \ldots \cup \tau(S_m)[K]) = \tau(E)[K]$ |
| **n-ary relationship** $R(E_1, \ldots, E_n)$, $n \geq 1$ | **Table** $\tau(R)$ |
| $X = \mathsf{attrs}(R)$ | |
| $K_i = \mathsf{id}(E_i)$, for $i = 1, \ldots, n$ | columns: $K_1 \ldots K_n X$ |
| if $R$ is functional on $E_i$ side | $key(\tau(R), K_i)$, for each $i = 1 \ldots n$ |
| if $R$ is non-functional | $key(\tau(R), K_1 \ldots K_n)$ |
| | $\tau(R)[K_i] \subseteq \tau(E_i)[K_i]$, for $i = 1, \ldots, n$ |
| if $R$ is total on $E_i$ side | $\tau(E_i)[K_i] \subseteq \tau(R)[K_i]$ |
| **N:M relationship** $R(E_1, E_2)$ | **Table** $\tau(R)$ |
| $X = \mathsf{attrs}(R)$ | columns: $K_1 K_2 X$ |
| $K_1 = \mathsf{id}(E_1)$, $K_2 = \mathsf{id}(E_2)$ | $key(\tau(R), K_1 K_2)$ |
| | $\tau(R)[K_i] \subseteq \tau(E_i)[K_i]$, for $i = 1, 2$ |
| if $R$ is total on $E_i$ side, $i = 1, 2$ | $\tau(E_i)[K_i] \subseteq \tau(R)[K_i]$ |
| **1:N relationship** $R(E_1, E_2)$ | **Option (a): Table** $\tau(R)$ |
| $X = \mathsf{attrs}(R)$ | columns: $K_1 K_2 X$ |
| $K_1 = \mathsf{id}(E_1)$, $K_2 = \mathsf{id}(E_2)$ | $key(\tau(R), K_1)$ |
| Let $R$ be functional on $E_1$ side | $\tau(R)[K_i] \subseteq \tau(E_i)[K_i]$, for $i = 1, 2$ |
| if $R$ is total on $E_i$ side, $i = 1, 2$ | $\tau(E_i)[K_i] \subseteq \tau(R)[K_i]$ |
| **1:N relationship** $R(E_1, E_2)$ | **Option (b): Add foreign key to** $\tau(E_1)$ |
| $K_1 = \mathsf{id}(E_1)$, $K_2 = \mathsf{id}(E_2)$ | |
| Let $R$ be functional on $E_1$ side | $\tau(E_1)[K_2] \subseteq \tau(E_2)[K_2]$ |
| if $R$ is total from $E_1$ to $E_2$ | $nonnull(\tau(E_1), K_2)$ |
| if $R$ is total from $E_2$ to $E_1$ | $\tau(E_2)[K_2] \subseteq \tau(E_1)[K_2]$ |
| **1:1 relationship** $R(E_1, E_2)$ | **Option (a): Table** $\tau(R)$ |
| $X = \mathsf{attrs}(R)$ | columns: $K_1 K_2 X$ |
| $K_1 = \mathsf{id}(E_1)$, $K_2 = \mathsf{id}(E_2)$ | $key(\tau(R), K_i)$, $i = 1, 2$ |
| | $\tau(R)[K_i] \subseteq \tau(E_i)[K_i]$, for $i = 1, 2$ |
| | $unique(\tau(R), K_2)$ |
| if $R$ is total on $E_i$ side, $i = 1, 2$ | $\tau(E_i)[K_i] \subseteq \tau(R)[K_i]$ |
| **1:1 relationship** $R(E_1, E_2)$ | **Option (b): Add foreign key to** $\tau(E_1)$ |
| $K_1 = \mathsf{id}(E_1)$, $K_2 = \mathsf{id}(E_2)$ | $\tau(E_1)[K_2] \subseteq \tau(E_2)[K_2]$ |
| | $unique(\tau(E_1), K_2)$ |
| if $R$ is total on $E_1$ side | $nonnull(\tau(E_1), K_2)$ |
| if $R$ is total on $E_2$ side | $\tau(E_2)[K_2] \subseteq \tau(E_1)[K_2]$ |

Table 7: Mapping ER model to relational model.

different relational schemas, e.g., there are choices in which direction a primary key of one table may be posted as a foreign key to another table while mapping a one-to-one relationship. Finally, we assume that the obtained table schemes were not merged (e.g., for reducing the number of tables, etc).

## 12.4 Conceptual schema extraction algorithms

Now we are ready to turn to the algorithm for extracting from a given relational database an ontology, described in terms of *DLR-DB* ontology language, plus the set of views, each of them corresponding to a unique construct in the ontology. The algorithm comprises of two steps:

1. Classify the relational tables in $\mathcal{R}$ according to the correlations between their keys and foreign keys.

2. Based on output of the first step, extract set of assertions in $\mathcal{K}$ of *DLR-DB* system and define the corresponding views.

To flesh out the above steps, we begin with the tables created by the standard design process. If a table is derived by the methodology of translating ER model to relational model, then Table 7 provides substantial knowledge about how the relational tables can be classified based on the appearance of their primary and foreign keys.

Before going into details let us fix up the general notation. Let $pk_i$ denote one of the keys of relational table $T_i$, i.e., a sequence of columns $\mathbf{A_i}$ such that $key(T_i, \mathbf{A_i})$[16]. Let $\mathsf{FK}_i$ be a set of all foreign keys of $T_i$, i.e., sequences of columns appearing on the left-hand side of the foreign key constraints of $T_i$, $\mathcal{F}(T_i)$. We denote with $fk_{i_j} \in \mathsf{FK}_i$ a foreign key of $T_i$ referencing table $T_j$. Additionally, if $\mathsf{Att}_i$ denotes all columns of $T_i$, then $\mathsf{X}_i = \mathsf{Att}_i - pk_i - \mathsf{FK}_i$.

The following function Classify classifies all tables in $\mathcal{R}$ into the three classes.

[16]In other words, $pk_i$ denotes primary key of $T_i$

**Function** Classify($\mathcal{R}$)

**Input**: Relational schema $\mathcal{R} = (\Psi, \Sigma)$.
**Output**: Sets $\Psi_B, \Psi_R, \Psi_S$ of classified tables.

$\Psi_B := \{\}$;
$\Psi_R := \{\}$;
$\Psi_S := \{\}$;
**foreach** $T_i \in \Psi$ **do**
  Let $card(X)$ be cardinality of set $X$;
  **if** $pk_i = \mathsf{FK}_i$ **and** $card(\mathsf{FK}_i) = 1$ **then**
    $\Psi_S := \Psi_S \cup T_i$;
  **end**
  **if** $pk_i \cap \mathsf{FK}_i = \emptyset$ **then**
    $\Psi_B := \Psi_B \cup T_i$;
  **end**
  **if** $pk_i \subseteq \mathsf{FK}_i$ **and** $card(\mathsf{FK}_i) > 1$ **then**
    $\Psi_R := \Psi_R \cup T_i$;
  **end**
**end**

Classify essentially attempts to reverse the function $\tau$. Looking to Table 7 we see that tables representing entities have keys disjoint with all their foreign keys where every foreign key represents functional binary relationship with another entity. Thus, tables of this type are classified as so-called *base tables*, denoted $\Psi_B$, and correspond to atomic formulae with a single component in *DLR-DB* $\mathcal{K}$. On the other hand, tables that directly represent relationships in ER have keys included in the set of all their foreign keys. We call tables of this type as *relationship tables*, denoted $\Psi_R$, which in *DLR-DB* correspond to atomic formulae with $n$ components. Finally, tables that represent sub-entities in ER have one key that coincide with their only foreign key. We call such tables as *specific tables*, denoted by $\Psi_S$, which correspond in our ontology to subsumption between two atomic formulas (both with single components).

We are, at this point, ready to define the algorithm SchemaExtract that takes as input a relational schema $\mathcal{R}$ and returns two structures as output. First, the set of *DLR-DB* assertions are generated, defining both atomic formulae and set of axioms between the formulae, thus expressing constraints in $\mathcal{K}$. Second, a view over the sources is defined for each atomic formula.

Informally, at step (a), the algorithm for every base table $T_i$ creates an atomic formula with a single component $R_i\,[pk_i]$ with the attributes in a one-to-one correspondence with key columns in $T_i$ (recall that according to Table 7 a base table corresponds to an entity), and defines a corresponding view by projecting on all non-foreign key columns of $T_i$. Once atomic formulae are generated, relations between objects denoted by these atomic formulae are identified. Roughly speaking, a foreign key in a base table $T_i$ determines the relationship between $T_i$ and its referenced table, say $T_j$. Thus, inclusion dependencies of the form $R\,[pk_i] \sqsubseteq R_i$[17] are generated stating that the component denoted by $pk_i$ ($2 \leq k \leq n$) of the relation $R$ is an instance of class $R_i$. Then the view is defined by joining the two

[17]When a relation has a single component, for the sake of simplicity the component name is omitted

**Algorithm** SchemaExtract($\mathcal{R}$).

**Input**: Relational schema $\mathcal{R} = (\Psi, \Sigma)$.
**Output**: *DLR-DB* system $\mathcal{S}$.
$\mathcal{A} := \{\}$;
$\mathcal{K} := \{\}$;
$\mathcal{V} := \{\}$;
Classify($\mathcal{R}$);
**(a) foreach** $T_i \in \Psi_B$ **do**
  $\mathcal{A} := \mathcal{A} \cup R_i\,[pk_i]$;
  $\mathcal{V} := \mathcal{V} \cup V_i : \pi_{pk_i, \mathsf{X}_i}(T_i)$;
**end**
**foreach** $T_i \in \Psi_B$ **do**
  **if** $\mathsf{FK}_i \neq \emptyset$ **then**
    **foreach** $fk_{i_j} \in \mathsf{FK}_i$ **do**
      $\mathcal{A} := \mathcal{A} \cup R_k\,[pk_i]$;
      $\mathcal{A} := \mathcal{A} \cup R_k\,[pk_j]$;
      $\mathcal{V} := \mathcal{V} \cup V_{i,j} : \pi_{pk_j, pk_i}(T_j \bowtie T_i)$;
      $\mathcal{K} := \mathcal{K} \cup R_k\,[pk_i] \sqsubseteq R_i$;
      $\mathcal{K} := \mathcal{K} \cup R_k\,[pk_j] \sqsubseteq R_j$;
      **if** $fk_{i_j}$ *is in* $\mathcal{N}(T_i)$ **then**
        $\mathcal{K} := \mathcal{K} \cup R_i \sqsubseteq R_k\,[pk_i]$;
      **end**
      **if** $fk_{i_j}$ *is in* $\mathcal{U}(T_i)$ **then**
        $\mathcal{K} := \mathcal{K} \cup$ (`funct` $R_k\,[pk_j]$);
      **end**
      **if** $fk_{i_j}$ *is in* $\mathcal{I}(T_i)$ **then**
        $\mathcal{K} := \mathcal{K} \cup R_j \sqsubseteq R_k\,[pk_j]$;
      **end**
    **end**
  **end**
**end**
**(b) foreach** $T_i \in \Psi_R$ **do**
  $\mathcal{V} := \mathcal{V} \cup V_i : \pi_{\mathsf{Att}_i}(T_i)$;
  **foreach** $fk_{i_j} \in \mathsf{FK}_i$ **do**
    $\mathcal{A} := \mathcal{A} \cup R_i\,[fk_{i_j}]$;
    $\mathcal{K} := \mathcal{K} \cup R_i\,[fk_{i_j}] \sqsubseteq R_j$;
    **if** $fk_{i_j} = pk_i$ **or** $fk_{i_j}$ *is in* $\mathcal{U}(T_i)$ **then**
      $\mathcal{K} := \mathcal{K} \cup$ (`funct` $R\,[fk_{i_j}]$);
    **end**
    **if** $fk_{i_j}$ *is in* $\mathcal{I}(T_i)$ **then**
      $\mathcal{K} := \mathcal{K} \cup R_j \sqsubseteq R_i\,[fk_{i_j}]$;
    **end**
  **end**
**end**
**(c) foreach** $T_i \in \Psi_S$ **do**
  $\mathcal{A} := \mathcal{A} \cup R_i\,[fk_{i_j}]$;
  $\mathcal{K} := \mathcal{K} \cup R_i \sqsubseteq R_j$;
  $\mathcal{V} := \mathcal{V} \cup V_i : \pi_{\mathsf{Att}_i}(T_i)$;
**end**
SetDisjCover($\mathcal{E}, \mathcal{C}$);
**return** $\mathcal{V}, \mathcal{A}, \mathcal{K}$;

**Function** SetDisjCover($\mathcal{E}, \mathcal{C}$).

**Input**: Sets of exclusion and covering dependencies, $\mathcal{E}(T_1, \ldots, T_m)$ and $\mathcal{C}(T_1, \ldots, T_m, T_0)$, resp.
**Output**: Sets of disjointness and covering assertions in $\mathcal{K}$.
**foreach** *exclusion dependency* $\mathcal{E}(T_1, \ldots, T_m)$ *in relational schema* **do**
  **foreach** $T_i, T_j$ *in* $\mathcal{E}(T_1, \ldots, T_m)$ **do**
    Let $T_i, T_j$ correspond to atomic formulae $R_i, R_j$ respectively;
    $\mathcal{K} := \mathcal{K} \cup R_i$ `disj` $R_j$;
  **end**
**end**
**foreach** *covering dependency* $\mathcal{C}(T_1, \ldots, T_m, T_0)$ *in relational schema* **do**
  Let $T_1, \ldots, T_m, T_0$ correspond to atomic formulae $R_i, \ldots, R_m, R_0$ respectively;
  $\mathcal{K} := \mathcal{K} \cup R_1, \ldots, R_m$ `cover` $R_0$;
**end**

tables $T_i$ and $T_j$ and by projecting on their keys. Whenever a foreign key in a base table $T_i$ is constrained to have non-null values, Table 7 suggests mandatory participation for class $R_i$ (that corresponds to table $T_i$) in a relation $R$. Thus, the inclusion assertion $R_i \sqsubseteq R\,[pk_i]$ is included in our ontology. On the other hand, if a foreign key of $T_i$ is constrained to take unique values, we deduce that a class denoted by relationship $R_j$ corresponding to the referenced table $T_j$ is functional, and a functional assertion (`funct` $R\,[pk_j]$) is associated to $\mathcal{K}$. Finally, a foreign key that participates in an inclusion dependency[18] clearly determines mandatory participation for class denoted by $R_j$ in a relation $R$, and thus the corresponding inclusion dependency is generated.

As for step (b) in SchemaExtract, the view is defined for each relationship table $T_i$ by projecting on all columns of $T_i$ (as already discussed before, relationship tables correspond directly to relationships). Then, relation $R$ is determined by defining the appropriate inclusion assertions (i.e., role-typing). Whenever we have a foreign key of $T_i$ coinciding with its key or a foreign key constrained to take unique values, we deduce (from Table 7) that $R$ is functional over the component which instances are of type $R_j$. Next, mandatory participation for class denoted by $R_j$ in $R$ is determined if the foreign key of $T_i$ occurs in the set of inclusion dependencies for $T_i$.

Finally, at step (c) inclusion assertions are generated for every specific table $T_i$, where a relation denoted by an atomic formula $R_i\left[fk_{i_j}\right]$ corresponding to $T_i$ is subsumed by relation denoted by an atomic formula $R_j\,[pk_j]$ corresponding to the referenced table $T_j$. Additionally, function SetDisjCover generates the set of disjointness and covering assertions induced by exclusion and covering dependencies, respectively.

The following example illustrates the extraction process.

**Example 12.1** *Suppose we are given the following table schemes:*
Employee*(*ssn*,* name*,* pcode*)* Project*(*code*,* title*)*
Permanent*(*essn*,* monthlySalary*)* Consultant*(*essn*,* dailyWage*)*

[18] We assume here that foreign key constraints $\mathcal{F}(T_i)$ are excluded from inclusion dependencies $\mathcal{I}(T_i)$.

*Assume the following constraints are expressed (we do not list key constraints, they are underlined in table schemes):*

*(1)* Employee [pcode] ⊆ Project [code]
*(2)* Permanent [essn] ⊆ Employee [ssn]
*(3)* Consultant [essn] ⊆ Employee [ssn]
*(4)* Project [code] ⊆ Employee [pcode]
*(5)* *nonnull*(Employee, pcode)
*(6)* Permanent [essn] ∩ Consultant [essn] = ∅
*(7)* Permanent [essn] ∪ Consultant [essn] = Employee [ssn]

*Applying the first step of the algorithm, tables* Employee *and* Project *will be classified as base tables (i.e., their key and foreign key columns are disjoint). Thus, at step (a) of the algorithm* SchemaExtract*, relationships having single components Employee and Project will be generated. Constraint (1) determines foreign key for table* Employee*. Thus, following the steps of our algorithm, the relationship AssignedTo*[19]*[employee, project] with two components is identified. At this point we define the following inclusion assertions (corresponding to role-typing): AssignedTo [employee] ⊑ Employee and AssignedTo [project] ⊑ Project. Furthermore, inclusion dependency in (4) determines mandatory participation for Project in a relationship AssignedTo [employee, project]. Thus, we obtain the inclusion assertion Project ⊑ AssignedTo [project]. On the other hand, nulls-not-allowed constraint in (5) imposes mandatory participation for Employee in AssignedTo [project], and we have the inclusion assertion Employee ⊑ AssignedTo [employee].*

*As for tables* Permanent *and* Consultant*, they will be classified by* Classify *as specific tables. Then, following step (c) of* SchemaExtract*, relationships with single components Permanent and Consultant will be identified, and the corresponding inclusion assertions Permanent ⊑ Employee and Consultant ⊑ Employee (based on inclusion dependencies in (2) and (3)) will be defined. Finally, constraints in (6) and (7) determine disjointness and covering assertions, and thus we have Permanent* `disj` *Consultant, and Permanent, Consultant* `cover` *Employee.*

## 12.5 Related Work

A potentially relevant work includes *database reverse engineering*, which is the process of two distinct steps: i) eliciting the data semantics from the system (like keys and foreign keys), and ii) expressing the extracted semantics with a high level data model, such as an ER diagram [Hai98]. Many approaches to this overlook the first step by assuming that some of the data semantics is known, but such assumptions may not hold for legacy databases (for a survey see [JS99]). Sophisticated algorithms and approaches to database reverse engineering have appeared in the literature over the years (e.g., [MM90, CBS94, Alh03]). Our ontology extraction algorithm uses the idea of classifying the relational tables based on correlations between their keys and foreign keys, which with a more broad view is defined in [CBS94].

[19] For the sake of clarity we use meaningful names for extracted relationships.

# 13 Query Containment and Satisfiability

## 13.1 Introduction

The query containment problem is a question of whether the result of one query is always contained in the result of another. The most interesting and practically significant instance of the problem arises when queries are posed against databases that satisfy constraints. In this case, query results that are otherwise not contained within each other might become contained if we restrict the attention to databases that satisfy a given set of constraints.

In dealing with this problem, an important issue is the form of the constraints and where they are coming from. If no restriction on the form of the constraints is placed, the containment problem may be computationally hard or even undecidable. However, in practice, constraints typically come from design tools that follow certain methodology, such as the Entity-Relationship Model.

We take the same approach and considers the constraints that typically arise from object-oriented design, in a single ontology setting. The specific data model that we use comes from F-logic, a knowledge representation formalism that has generated considerable interest in the academia, within various standardization efforts, and commercially as a means for building ontologies and for reasoning on the Semantic Web. F-logic queries have one property that is not present in the query classes considered so far: it has *meta-querying* capability, i.e., it can query data *and schema* in a uniform way. This property is considered important in knowledge integration and service discovery on the Semantic Web. The need to access schema information has been recognized in other languages as well, albeit the facilities that are made available to the user are often rather awkward. For instance, SQL databases provide access to the system catalog and Java has reflection API for the same purpose.

Here, we show that query containment is decidable for a subset of conjunctive F-logic meta-queries, and is in NP. This subset, which we call *F-logic lite*, excludes negation and default inheritance, and allows only a limited form of cardinality constraints.

Fortunately, for this contribution we need only a very limited amount of background on F-logic, which will be introduced in this section. However, we assume familiarity with Datalog and Logic Programming.

**Basic F-logic notation.** Unlike classical predicate calculus, F-logic has special atomic formulas to represent the various object-oriented concepts that are common to object-oriented and frame-based systems. For instance, `john:student` states that object `john` is a member of class `student`; `freshman::student` and `student::person` state that class `freshman` is a subclass of the class `student` and `student` is a subclass of `person`. These statements imply, for instance, that `john:person` (`john` is a student) and `freshman::person` (class `freshman` is a subclass of `person`) are true statements.

A statement of the form `john[age->33]` means that object `john` has an attribute, `age`, whose value is `33`. Actually, this really means that `33` is just *one of the values* of the attribute `age` — to say that `33` is the *only* value, one would need a cardinality constraint, as explained later.

Common constraints such as *type constraints* and *cardinality constraints* are specified via so called *signature* statements. A typical signature has the form

`person[age*=>number]`. It says that the attribute `age` of class `student` has the *type* `number` and that this type is inherited by subclasses and class instances of `person`. This acts as a constrain on the statements of the form `john[age->33]`. That is, for every object in class `person` the value of the attribute `age` must be an object of class `number`.

Cardinality constraints can also be specified using signature statements. For instance, to say that the attribute `age` has at most one value, one would write `person[age {0:1} *=> number]`. Another frequently used cardinality constraint states that a certain attribute in a class is `mandatory`, i.e., it must have at least one value on any object in the class. For instance, to say that the `name` attribute is mandatory in class `person`, we write `person[name {1:*} *=> string]`.

As mentioned in the introduction, F-logic treats object data and meta-data in a uniform way. This is primarily manifested in the following two ways:

- Classes are also objects, so, for example, statements like `student:class` are correct. Here `student`, which was previously seen in a context of a class (`john:student`), now occurs in a context of an object—a member of the class `class`. (Note that it *does not* follow from this and the previous statements that `john:class`, `freshman:class`, or `student::class`.)

- Variables can occur anywhere an object, an attribute, or a class is allowed. In the following, we will be using capitalized words as variable names. For instance, `john:X`, `Y::person`, `john[Att->33]`, and `person[Att*=>Val]` are all allowed statements where $X$, $Y$, $Att$, and $Val$ are variables.

The above properties enable simple and natural formulation for a wide variety of meta-queries—queries that return information about the schema of the database instead of the data stored in it. For instance, the answer to the meta-query

```
?- X::person.
```

could be `X = employee` and `X = student`, and the answer to the meta-query

```
?- student[Att*=>string].
```

could be `Attr = name` and `Attr = major`. Queries can also mix the meta and the object levels. For instance, the mixed query (where "," denotes "and")

```
?- student[Att*=>string], john[Att->Val].
```

will return a set of attribute-value pairs for the attributes of type `string` in class `student`. Only the attributes that have defined values for object `john` are returned. Incidentally, `john` does not need to be a member of class `student` for such a query to return a result.

**Examples of meta-queries.** Meta-querying is a commonly acknowledged need in knowledge representation — especially on the Semantic Web. To show that it is not such an esoteric idea, we give some examples of meaningful meta-queries and the corresponding query containments. Consider the following rule:

```
q(A,B) :- T1[A*=>T2], T2::T3, T3[B*=>_].
```

As usual, `:-` here denotes Datalog implication. The part to the left of `:-` is the *head* of the rule and the part to the right is the rule `body`. The comma separating the statements in the rule body is an abbreviation for conjunction, and the symbol `_` is a common notation for a completely new variable that does not appear anywhere else in the rule. (Different occurrences of `_` denote different variables.)

The above rule defines a set of pairs `(A,B)` of attributes that are *joinable* in a path expression of the form `A.B` (that is, the range of `A` is contained in the domain of `B`). If we now examine the following rule

```
qq(A,B) :- T1[A*=>T2], T2[B*=>_].
```

we will see that the query containment `q` $\subseteq$ `qq` holds.

For another example, consider the following rule:

```
q(Att,Class,Type) :-
     Class[Att {1,*} *=> _],
     Class[Att*=>Type],
     _:Class.
```

Recall that `{1,*}` is a cardinality constraint that says that `Att` must have at least one value, i.e., it is a *mandatory* attribute. This rule defines a set of triples `(Att,Class,Type)` such that `Att` is a mandatory attribute in `Class` of type `Type` and, furthermore, `Class` is nonempty. Note that here we are querying meta-data that is subject to certain constraints: only the mandatory attributes are of interest and only the classes that have at least one member in the database. Consider now the following rule:

```
qq(Att,Class,Type) :-
      O:Class, O[Att->V], V:Type.
```

It is easy to see that the containment `q` $\subseteq$ `qq` holds.

**Low-level encoding of F-logic primitives.** The above notation was used to motivate the problem and to define the target class of queries. The actual theoretical development will use an encoding of the semantics of a subset of F-logic using the standard logic programing notation. The encoding relies on the equivalence result of [KLW95].

The subset of F-logic, which we will consider will be referred to as *F-logic Lite*. This subset is characterized by the absence of negation and nonmonotonic features of F-logic (such as default inheritance) and by allowing only the cardinality constraints of the form `{1:*}` — a constraint that says that the corresponding attribute in mandatory — and of the form `{0:1}` — a constraint that marks functional (single-valued) attributes. Some other features, such as default values and non-inheritable types, are also ignored.

The encoding, uses the following predicates, whose set we will denote by $P_{FL}$:

- $\mathsf{member}(O, C)$: object $O$ is a member of class $C$.
  This is the encoding for $O : C$.

- $\mathsf{sub}(C_1, C_2)$: class $C_1$ is a subclass of class $C_1$.
  This encodes the statement $C_1 :: C_2$.

- $\mathsf{data}(O, A, V)$: attribute $A$ has value $V$ on object $O$. This is the encoding for $O[A\texttt{->}V]$.
- $\mathsf{type}(O, A, T)$: attribute $A$ has type $T$ for object $O$ (recall that in F-logic classes are also objects). This encodes the statements of the form $O[A\texttt{*=>}T]$.
- $\mathsf{mandatory}(A, O)$: attribute $A$ is mandatory for object (class) $O$, i.e., it must have at least one value for $O$. This is an encoding of statements of the form $O[A\,\{\texttt{1:*}\}\texttt{*=>\_}]$.
- $\mathsf{funct}(A, O)$: $A$ is a functional attribute for the object (class) $O$, i.e., it can have at most one value for $O$. This statement encodes $O[A\,\{\texttt{0:1}\}\texttt{*=>\_}]$.

Note that this encoding places meta-data (classes and attributes) and object data at the same level, which is needed for supporting F-logic meta-queries. This encoding is also related to, but is slightly different from, the usual encoding of RDF in first-order logic.

We can now formulate the axioms that represent the low-level encoding of the F-logic primitives discussed above in standard predicate notation. We annotate each rule in the encoding to make it easier to follow.

1. *Type correctness:*
   $\mathsf{member}(V, T) \leftarrow \mathsf{type}(O, A, T), \mathsf{data}(O, A, V)$.
   This encodes the semantics of the constraint that an F-logic signature like `O[A *=> T]` imposes on a statement like `O[A -> V]`; that is, that `V` must be of type `T`.

2. *Subclass transitivity:*
   $\mathsf{sub}(C_1, C_2) \leftarrow \mathsf{sub}(C_1, C_3), \mathsf{sub}(C_3, C_2)$.
   This rule encodes the fact that the subclass relationship is transitive.

3. *Membership property:*
   $\mathsf{member}(O, C_1) \leftarrow \mathsf{member}(O, C), \mathsf{sub}(C, C_1)$.
   This is the usual property that relates class membership and subclass relationship: `O:C` and $\texttt{C::C}_1$ imply $\texttt{O:C}_1$.

4. *Functional attribute property:*
   $V = W \leftarrow$
   $\mathsf{data}(O, A, V), \mathsf{data}(O, A, W), \mathsf{funct}(A, O)$.
   This rule states that if we have `O[A->V]`, `O[A->W]`, and the attribute `A` is single-valued then $X$ must equal $W$.

5. *Mandatory attributes definition:*
   $\forall O, A\, \exists V\ \mathsf{data}(O, A, V) \leftarrow \mathsf{mandatory}(A, O)$. This states that mandatory attributes must have at least one value. Note that this is *not* a Datalog rule (not even a Horn rule) because it has an existentially quantified variable in the head.

6. *Inheritance of types from classes to members:*
   $\mathsf{type}(O, A, T) \leftarrow \mathsf{member}(O, C), \mathsf{type}(C, A, T)$.
   This rule expresses the usual property of type inheritance: the type of an attribute is inherited from superclasses to class members.

7. *Inheritance of types from classes to subclasses:* $\mathsf{type}(C, A, T) \leftarrow \mathsf{sub}(C, C_1), \mathsf{type}(C_1, A, T)$.
   This states that subclasses inherit types from superclasses.

8. *Supertyping:*
   $\mathsf{type}(C, A, T) \leftarrow \mathsf{type}(C, A, T_1),\ \mathsf{sub}(T_1, T)$.
   This states that `T1::T` and `C[A *=> T1]` entails `C[A *=> T]`. This is also one of the usual properties of typing: if an attribute has certain type then any supertype of that type will also do.

9. *Inheritance of mandatory attributes to subclasses:*
   $\mathsf{mandatory}(A, C) \leftarrow \mathsf{sub}(C, C_1), \mathsf{mandatory}(A, C_1)$.
   This states that a mandatory attribute of a class is also a mandatory attribute of its subclasses.

10. *Inheritance of mandatory attributes from classes to their members:*
    $\mathsf{mandatory}(A, O) \leftarrow$
    $\mathsf{member}(O, C), \mathsf{mandatory}(A, C)$.
    Like in (9), but this time inheritance of the mandatory property is to class members rather than subclasses.

11. *Inheritance of functional property to subclasses:*
    $\mathsf{funct}(A, C) \leftarrow \mathsf{sub}(C, C_1),\ \mathsf{funct}(A, C_1)$.
    If `A` is a single-valued attribute in a class then it must be single-valued in the subclasses of that class.

12. *Inheritance of functional property to members:*
    $\mathsf{funct}(A, O) \leftarrow \mathsf{member}(O, C), \mathsf{funct}(A, C)$.
    Like (11), but this time inheritance of the single-valued property is to class members.

In the following, we will denote the above set of rules by $\Sigma_{FL}$. We will also refer to the $i$-th rule as $\rho_i$. The above statements are all Datalog rules except $\rho_4$ and $\rho_5$. Rule $\rho_4$ uses equality in the head and $\rho_5$ has an existential quantifier in the rule head and thus invents new (fresh) values. Also, most of the rules are recursive. Additional properties of this encoding are studied in the next section.

Thanks to the above encoding, we can express any F-logic Lite database and its schema as a relational database augmented with a set of rules for deriving new information and for expressing constraints. We shall consider *only* the databases that satisfy the above set of rules.

**Query containment.** The query containment problem for F-logic Lite can be now stated as follows. Given a pair of meta-queries, $q_1$ and $q_2$, over the predicates $P_{FL}$ of the above encoding of F-logic Lite, we say that $q_1$ *is contained in* $q_2$ with respect to $\Sigma_{FL}$, denoted $q_1 \subseteq_{\Sigma_{FL}} q_2$, if for every database $B$ that satisfies $\Sigma_{FL}$ we have $q_1(B) \subseteq q_2(B)$, where $q(B)$ denotes the result of query $q$ on $B$.

In our case we will focus on positive *conjunctive queries* [AHV95], i.e., the queries that are conjunctions of the predicates in $P_{FL}$ and no negation is allowed.

## 13.2 Checking Query Containment

In this section we study the problem of query containment for queries expressed over the schema $P_{FL}$ derived from the low-level encoding of F-logic Lite. We recall that the encoding is entirely relational plus the rules $\Sigma_{FL}$ shown in Section 13.1. First, we introduce the notion of *homomorphism*, which is of fundamental importance in conjunctive query containment [CM77]. We remind that for conjunctive queries over generic relational schemata without constraints, a homomorphism from the body of a query $q_2$ to the body of another query, $q_1$, which also maps the head of $q_2$ to the head of $q_1$, implies $q_1 \subseteq q_2$. Indeed, if the body of $q_1$ is mapped by a homomorphism to facts of a database, $B$, so that the head of $q_1$ is mapped to tuple $t$, then by composing the homomorphisms $q_2 \longrightarrow q_1$ and $q_1 \longrightarrow B$ we get a homomorphism that maps the body of $q_2$ to the facts of $B$ and the head of $q_2$ to $t$. Thus, every tuple produced by $q_1$ is also produced by $q_2$.

Given a database $B$ and a query $q$, a *homomorphism* from $q$ to $B$ is a function from the symbols of $q$ to the values of $B$ that maps every constant occurring in $q$ to itself and variables of $q$ to constants in $B$. This induces a well defined map from the conjuncts of $q$ to the tuples of the *corresponding* relations in $B$. In particular, a conjunct $r(c_1, \ldots, c_n)$ in $q$ is mapped by a homomorphism $\mu$ to a fact $r(\mu(c_1), \ldots, \mu(c_n))$ in $B$. We also extend the definition to sets of conjuncts: given a set of conjuncts $C = \{c_1, \ldots, c_n\}$, we define $\mu(C) = \{\mu(c_1), \ldots, \mu(c_n)\}$.

### 13.2.1 Containment by chasing

Now we come to the notion of *chase* of a query with respect to our set of rules $\Sigma_{FL}$. The chase [MMS79, JK84] is a tool for representing databases that satisfy certain dependencies, and it is used to check implication of dependencies and containment of queries under dependencies. Given a database, the chase is constructed by a sort of *repair* of the database w.r.t. the rules that are not satisfied. In particular, in our case, violations of all rules except $\rho_4$ are repaired with the *addition* of suitable tuples, while violations of $\rho_4$ are repaired by *equating* constants that are not equal. The rules in $\Sigma_{FL} - \{\rho_4\}$ are called *tuple-generating dependencies*, while $\rho_4$ is an *equality generating dependency*. The query $q$ to be "chased" is treated as a database, and new tuples (or conjuncts) are added according to the rules. The chase of a query $q$, denoted $chase_{\Sigma_{FL}}(q)$, is a database constructed starting from $body(q)$ [CK06]. The construction of the chase proceeds iteratively applying the rules; in case of cyclic rules, which is our case, it is possible that the construction of the chase does not terminate.

**Example 13.1** Consider the following meta-query:

$$\begin{aligned} q(V_1, V_2) \quad \leftarrow \quad & \mathsf{data}(O, A, V_1), \mathsf{data}(O, A, V_2), \\ & \mathsf{funct}(A, C), \mathsf{member}(O, C) \end{aligned}$$

In the construction of $chase_{\Sigma_{FL}}(q)$, rule $\rho_{12}$ will add the conjunct $\mathsf{funct}(A, O)$ and then, by rule $\rho_4$, we will replace $V_2$ with $V_1$ and obtain

$$\begin{aligned} q(V_1, V_1) \quad \leftarrow \quad & \mathsf{data}(O, A, V_1), \mathsf{data}(O, A, V_1), \\ & \mathsf{funct}(A, O), \\ & \mathsf{funct}(A, C), \mathsf{member}(O, C) \end{aligned}$$

Therefore, the chase procedure may have side effects on the head of the query. Henceforth we shall use $head(chase_{\Sigma_{FL}}(q))$ to denote the head of the query $q$ as it is transformed by the construction of $chase_{\Sigma_{FL}}(q)$ according to $\Sigma_{FL}$.

A notion that is needed for technical reasons, and also for somehow "measure" how far we go in the construction of the chase, is the *chase graph*. In this graph, the notion of *level* roughly indicates how far we need to go in the chase starting from the initial query. This is crucial for our purposes, since we will show that in order to test containment we will need to examine the chase only up to a certain level. Intuitively, the initial facts are considered at level 0, and at every iterative construction step that starts from level $k$, a new level, numbered with $k + 1$, is created.

The following theorem provides the basic tool for checking containment of queries over $P_{FL}$. Intuitively, in the containment check between two queries $q_1$ and $q_2$, $body(q_1)$ represents the set of tuples in a generic database $B$ that lead to an answer tuple in $q(B)$. Since we need to check containment under $\Sigma_{FL}$, we need to take into account not only $body(q_1)$, but also further tuples that the rules guarantee to be in $B$; therefore, we need to consider $chase_{\Sigma_{FL}}(q_1)$. This is stated as follows.

**Theorem 13.2** *Let $q_1$ and $q_2$ be two conjunctive queries over $P_{FL}$ with the same arity. Then $q_1 \subseteq_{\Sigma_{FL}} q_2$ if and only if there exists a homomorphism that sends the conjuncts of $body(q_2)$ to conjuncts in $chase_{\Sigma_{FL}}(q_1)$ and $head(q_2)$ to $head(chase_{\Sigma_{FL}}(q_1))$.*

The previous theorem establishes a criterion for checking containment, but it is not directly applicable, since it does not suggest an algorithm for deciding containment. In fact, the iterative construction of the chase by application of the chase rules may not terminate. In the following section we will first show some properties of the chase, and then we will show that, in order to test containment of meta-queries, only a finite portion of the chase is necessary.

### 13.2.2 Decidabilty of containment

In this section we show that containment of object meta-queries is decidable, and we give a nondeterministic polynomial time algorithm for checking containment between two object meta-queries. We show that only a finite portion of the possibly infinite chase is necessary to check containment with the technique suggested by Theorem 13.2, and then we present an algorithm for query containment, showing an upper bound for the complexity of the problem.

For technical reasons, we will do the chase in a special way. This will isolate some of the peculiarities of the chase and allow us to concentrate on more important properties. Namely, we shall first proceed with the chase with respect to all the rules except for $\rho_5$; such a preliminary chase always terminates, since no new constant is generated.

To simplify matters, we will view all tuples in $chase_{\Sigma^-_{FL}}(q)$ as being at level 0, where $\Sigma^-_{FL} = \Sigma_{FL} - \{\rho_5\}$. This will allow us to isolate the initial part of the chase from the cyclic phase (if the latter takes place). First of all it is not difficult to notice that, in the construction of the chase for a query $q$ with respect to the set $\Sigma_{FL}$, the only way to have an infinite chase is the iterative application of rules $\rho_5$-$\rho_1$-$\rho_6$-$\rho_{10}$. This happens when $q$ contains at least a set of atoms specifying a cycle of mandatory attributes $A_1, \ldots, A_k$

belonging to classes $T_1, \ldots, T_k$, respectively, where $A_i$ is of type $T_{i+1}$ for all $i \in \{1, \ldots, k-1\}$ and $A_k$ is of type $T_1$. More precisely, we need $q$ to have conjuncts of the following form:

$$\begin{array}{l}
\mathsf{mandatory}(A_1, T_1) \\
\mathsf{type}(T_1, A_1, T_2) \\
\ldots \\
\mathsf{mandatory}(A_{k-1}, T_{k-1}) \\
\mathsf{type}(T_{k-1}, A_{k-1}, T_k) \\
\mathsf{mandatory}(A_k, T_k) \\
\mathsf{type}(T_k, A_k, T_1)
\end{array}$$

In such a case, if there is no atom in $q$ of the form $\mathsf{data}(T_1, A_1, v)$, where $v$ is a constant or variable, the chase process yields the following series of conjuncts:

$$\begin{array}{rl}
\text{cycle } 1: & \mathsf{data}(T_1, A_1, v_1) \\
& \mathsf{member}(v_1, T_2) \\
& \mathsf{type}(v_1, A_2, T_3) \\
& \mathsf{mandatory}(A_2, v_1) \\
\\
\text{cycle } 2: & \mathsf{data}(v_1, A_2, v_2) \\
& \mathsf{member}(v_2, T_3) \\
& \mathsf{type}(v_2, A_3, T_4) \\
& \mathsf{mandatory}(A_3, v_2) \\
& \ldots \\
& \ldots \\
\text{cycle } k-1: & \mathsf{data}(v_{k-2}, A_{k-1}, v_{k-1}) \\
& \mathsf{member}(v_{k-1}, T_k) \\
& \mathsf{type}(v_{k-1}, A_k, T_1) \\
& \mathsf{mandatory}(A_k, v_{k-1}) \\
\\
\text{cycle k}: & \mathsf{data}(v_{k-1}, A_k, v_k) \\
& \mathsf{member}(v_k, T_1) \\
& \mathsf{type}(v_k, A_1, T_2) \\
& \mathsf{mandatory}(A_1, v_k)
\end{array}$$

In the rest of the chase, at levels greater than 0, the only other applications of a rule in $\Sigma_{FL}$ occur due to the application of $\rho_3$ or $\rho_8$. In this case, depending on the conjuncts al level 0, new cycles may start. All other rules are applied in $chase_{\Sigma^-_{FL}}(q)$ (i.e., in the initial construction of level 0 of the chase graph), and they are never applied again at higher levels.

The aforementioned properties of the chase disclose that the chase has some regularity properties; indeed, such properties can be exploited for showing, analogously to what is done in [JK84], that the chase has a sort of periodicity, so that the levels above a certain limit provide no additional information regarding containment. This is the main property behind the key result, mentioned earlier, asserting that a set of $n$ conjuncts in the chase

of a query $q$ can be mapped by a homomorphism to another set of conjuncts at levels lower than a certain limit, which depends on $n$ and $q$.

**Lemma 13.3** *Given a query $q$ over $P_{FL}$, consider a set of conjuncts $C = \{c_1, \ldots, c_n\}$ in $chase_{\Sigma_{FL}}(q)$. Then there exists a homomorphism $h$ from $C$ to $chase_{\Sigma_{FL}}(q)$ such that for every $i$, $1 \leq i \leq n$, we have $level(h(c_i)) \leq n \cdot \delta$, where $\delta = 2 \cdot |q|$.*

As a consequence of the previous lemma, to check $q_1 \subseteq_{\Sigma_{FL}} q_2$ we only need to find a homomorphism from $q_2$ to an initial, a priori bounded finite segment of $chase_{\Sigma_{FL}}(q_1)$.

**Theorem 13.4** *Let $q_1$ and $q_2$ be two conjunctive queries over $P_{FL}$ with the same arity. Then $q_1 \subseteq_{\Sigma_{FL}} q_2$ if and only if there exists a homomorphism that sends the conjuncts of $body(q_2)$ to conjuncts in the first $|q_2| \cdot \delta$ levels of $chase_{\Sigma_{FL}}(q_1)$, and $head(q_2)$ to $head(chase_{\Sigma_{FL}}(q_1))$, where $\delta = 2 \cdot |q_1|$.*

Finally, we characterize the computational complexity of the problem of checking containment of queries by giving an upper bound for the problem. We do this by exhibiting an algorithm for checking containment, which obeys the bounds.

**Theorem 13.5** *Consider two conjunctive queries $q_1, q_2$ on $P_{FL}$. Containment $q_1 \subseteq_{\Sigma_{FL}} q_2$ of $q_1$ in $q_2$ can be decided by a nondeterministic algorithm running in time polynomial in $|q_1|$ and $|q_2|$.*

## 13.3 Related work

The problem of query containment has attracted considerable interest in the database and knowledge representation communities. In databases, query containment is key to query optimization and schema integration [ASU79b, JK84, MLF00], and in knowledge representation it has been widely used in Description Logic [BCM+03a] for object classification, schema integration, service discovery, and more [CDGL02, LH03]. A study of this problem was pioneered by Johnson and Klug in [JK84] in the relational case, where functional and inclusion dependencies hold on the database schema, and then further studied in other works [CDGL98a, Cal06]. A study about query containment over object databases is found in [LS97]; however, in the framework of this work there is no possibilities of meta-queries as in F-Logic.

F-logic was introduced in [KL89, KLW95] as a formalism for object-oriented deductive databases. Since then it received further development and was implemented in the FLORA-2 and FLORID knowledge representation systems as well as commercially [YKZ03, FLOa, FHL+98, FLOb, Ont, SD02]. We considered the problem of query containment for conjunctive meta-queries over F-logic knowledge bases; this important class of queries has not been covered by the known results on query containment. Earlier works on the subject are [CDGL98a, Cal06]; F-logic queries and the associated constraints are different from the formalisms adopted in these works: for instance, decidability does not follow from [CDGL98a] because conjunctive F-logic queries involve certain recursion, unions, and joins of ternary predicates, which are not allowed in [CDGL98a].

The practical upshot of the results we presented here is that they pave the way to the use of query containment for F-logic based applications in query processing, ontology integration, and Web service modeling and discovery [AL04, Kif05, LBT92, dB05, BBB+05]. These results are also relevant to the Semantic Web language RDF [Le99] and the recently proposed query language for it, called SPARQL [PS05]; RDF has many of the meta-data features of F-logic and SPARQL can query them.

# 14 Query Satisfiability, Disjointness, and Containment

## 14.1 Introduction

Query management is very significative for ontology design and maintenance, since we can make use of queries to construct views that can be exploited for the task of view-based ontology design and maintenance. More precisely, query management groups the following reasoning techniques

- *Query satisfiability*, i.e., determining if a query $q$ is consistent with respect to the ontology over which it is formulated, i.e., establishing whether there exists a model of the ontology in which the query is evaluated to a non-empty set.
- *Query disjointness*, i.e., determining whether two queries expressed over the same ontology are disjoint from each other. They are disjoint if their answer sets are always mutually exclusive for every possible model of the ontology.
- *Query containment*, i.e., determining whether a query $q_1$ expressed over an ontology is contained in a query $q_2$ expressed over the same ontology, i.e., establishing whether in all possible models of the ontology, $q_1$ is evaluated to be a subset of the evaluation of $q_2$.

Notice that the above techniques are interesting in all setting considered by the TONES logical framework, i.e., stand-alone ontologies, situated ontologies, or peer ontologies. For easy of exposition, we consider them here in the setting of stand-alone ontology.

Techniques for query management have been studied under different choices for the language used to specify the ontology over which the query are issued and the language used to express queries (see e.g. [CDGL97, CDGL98a, CDGL07, HT00]). We consider here this problem in the setting in which

(*a*) ontologies are expressed in *DL-Lite* [CDGL$^+$05a], a DL that is specifically tailored to capture basic ontology languages, while keeping low complexity of reasoning (in particular, in LogSpace in the size of the instances in the knowledge base, a.k.a. *data complexity*), and therefore it is particularly suited for reasoning services relying on queries;
(*b*) queries belong to the class of conjunctive queries that is the most expressive class of queries that go beyond instance checking, and for which decidability of query answering (and therefore query containment) has been proved in DLs [CDGL00, OCE06, GHLS07].

*DL-Lite* is a subset of the DL $\mathcal{SHOIQ}$, introduced in Section 2. In particular, *DL-Lite* presents some limitations in using constructs to specify complex concepts and in the kind of concepts that can occur in the right-hand of inclusion between concepts; also *DL-Lite* does not allow for the specification of inclusions between roles, and permits to express only functionality restrictions on roles or on the inverse of roles, but does not allow for generic cardinality constraints. All the above limitations ensure the nice

computational complexity characteristics we mentioned before, i.e., reasoning services, as query answering of (union of) conjunctive queries, are in LogSpace in data complexity. Notably, query answering in *DL-Lite* can be solved by means of evaluation of suitable domain independent first-order logic queries (thus expressible in SQL) over the underlying *DL-Lite* ABox considered as a flat relational database. This allows us to take advantage of well established Relational Data Base Management System (RDBMS) technology for reasoning over queries in *DL-Lite*.

We point out that, although *DL-Lite* is quite simple from the language point of view, it allows for querying the extensional knowledge of a KB in a much more powerful way than usual DLs, in which only membership to a concept or to a role can be asked, whereas *DL-Lite* allows for using conjunctive queries of arbitrary complexity. At the same time, *DL-Lite* is able to capture the main notions (though not all, obviously) of both ontologies, and of conceptual modeling formalisms used in databases and software engineering (i.e., ER and UML class diagrams). In particular, *DL-Lite* assertions allow us to specify *ISA*, e.g., stating that a concept is subsumed by another concept; *disjointness*, e.g., stating that two concepts are disjoint; *role-typing*, e.g., stating that the first or the second component of a relation is an instance of a certain concept; *participation constraints*, e.g., stating that all instances of a concept participate to a relation as the first or the second component; *non-participation constraints*, e.g., stating that no instance of a concept participates to a relation as the first or the second component; *functionality restrictions* on relations. Notice that DL-Lite is a strict subset of OWL Lite[20], which presents some constructs (e.g., some kinds of role restrictions) that are non expressible in DL-Lite, and that make reasoning in OWL Lite non-tractable in general.

In the rest of this section we first formally present the *DL-Lite* language, then we tackle query satisfiability, query disjointness and query containment. Finally, briefly discuss some related work.

## 14.2 DL-Lite

As usual in DLs, *DL-Lite* allows for representing the domain of interest in terms of concepts and roles. *DL-Lite* concepts are defined as follows:

$$\begin{array}{rcl} B & ::= & A \mid \exists r \mid \exists r^- \\ C & ::= & B \mid \neg B \end{array}$$

where $A$ denotes an atomic concept and $r$ denotes an (atomic) role; $B$ denotes a *basic concept* that can be either an atomic concept, a concept of the form $\exists r$, or a concept of the form $\exists r^-$, which involves an *inverse role*. $C$ denotes a (general) concept. Note that we use negation of basic concepts only, and we do not allow for disjunction.

As said in Section 2, a DL KB is a triple $(\mathcal{T}, \mathcal{H}, \mathcal{A})$, where $\mathcal{T}$ is a TBox, $\mathcal{H}$ a role hierarchy, and $\mathcal{A}$ an ABox. Since *DL-Lite* does not allow for specifying subsumptions between roles, a *DL-Lite* KB is simply a pair $(\mathcal{T}, \mathcal{A})$. In particular, *DL-Lite TBox* assertions are of the form

| | |
|---|---|
| $B \sqsubseteq C$ | *inclusion assertions* |
| $(\mathsf{funct}\ r), (\mathsf{funct}\ r^-)$ | *functionality assertions* |

[20] http://www.w3.org/TR/owl-features

An inclusion assertion expresses that a basic concept is subsumed by a general concept, while a functionality assertion expresses the (global) functionality of a role, or of the inverse of a role.

We observe that we might include $B_1 \sqcup \mathcal{B}_2$ in the constructs for the left-hand side of the inclusion assertions (where $\sqcup$ denotes union) and $C_1 \sqcap C_2$ in the constructs for the right-hand side (where $\sqcap$ denotes conjunction). In this way, however, we would not extend the expressive capabilities of the language, since these constructs can be simulated by considering that $B_1 \sqcup B_2 \sqsubseteq C$ is equivalent to the pair of assertions $B_1 \sqsubseteq C$ and $B_2 \sqsubseteq C$, and that $B \sqsubseteq C_1 \sqcap C_2$ is equivalent to $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$. Similarly, we might add $\bot$ (denoting the empty set) to the constructs for the left-hand side and $\top$ (denoting the whole domain) to those for the right-hand side.

Hereinafter, we call *positive inclusions (PIs)* assertions of the form $B_1 \sqsubseteq B_2$, whereas we call *negative inclusions (NIs)* assertions of the form $B_1 \sqsubseteq \neg B_2$.

As for the ABox, *DL-Lite* allows for assertions of the form:

$$B(a), r(a, b) \quad \textit{membership assertions}$$

where $a$ and $b$ are constants. These assertions state respectively that the object denoted by $a$ is an instance of the basic concept $B$, and that the pair of objects denoted by $(a, b)$ is an instance of the role $r$.

As already said, *DL-Lite* allows for querying the extensional knowledge of a KB by means of conjunctive queries. A *conjunctive query (CQ)* over a *DL-Lite* knowledge base $\mathcal{K}$ is an expression of the form

$$\{ \vec{x} \mid \exists \vec{y}.\mathit{conj}(\vec{x}, \vec{y}) \}$$

where $\vec{x}$ are the so-called *distinguished variables*, $\vec{y}$ are existentially quantified variables called the *non-distinguished* variables, and $\mathit{conj}(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $B(z)$, or $r(z_1, z_2)$, where $B$ and $r$ are respectively a basic concept and a role in $\mathcal{K}$, and $z$, $z_1$, $z_2$ are constants in $\mathcal{K}$ or variables in $\vec{x}$ or $\vec{y}$. The number of distinguished variables in a query $q$ is called the *arity* of the $q$. A *boolean conjunctive query* is a CQ that does not involve distinguished variables, i.e., it is an expression of the form $\{ \mid \exists \vec{y}.\mathit{conj}(\vec{y}) \}$. Given a boolean CQ of the form above, we may simply denote it as $\exists \vec{y}.\mathit{conj}(\vec{y})$.

A *union of conjunctive queries (UCQ)* $q$ is a query of the form

$$\{ \vec{x} \mid \bigvee_{i=1,\ldots,n} \exists \vec{y_i}.\mathit{conj}_i(\vec{x}, \vec{y_i}) \}$$

where each $\mathit{conj}_i(\vec{x}, \vec{y_i})$ is, as before, a conjunction of atoms and equalities with free variables $\vec{x}$ and $\vec{y_i}$. Obviously, the class of union of conjunctive queries contains the class of conjunctive queries. The notion of boolean UCQ is analogous to the notion of boolean CQ.

As for the semantics of a *DL-Lite* KB, we refer the reader to Section 2 for the notion of interpretation of concepts, roles, membership assertions, and TBox inclusion assertions. Furthermore, we say that an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies a functionality assertion $(\mathsf{funct}\ r)$ if $(c, c') \in r^{\mathcal{I}} \wedge (c, c'') \in r^{\mathcal{I}} \supset c' = c''$, similarly for $(\mathsf{funct}\ r^-)$. We also say that

an interpretation $\mathcal{I}$ for a *DL-Lite* KB $\mathcal{K}$ is a *model* of $\mathcal{K}$, denoted $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ satisfies all inclusion, functionalities, and membership assertions in $\mathcal{K}$.

A conjunctive query $q = \{ \vec{x} \mid \exists \vec{y}.conj(\vec{x}, \vec{y}) \}$ is interpreted in an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as the set $q^{\mathcal{I}}$ of tuples $\vec{c} \in \Delta^{\mathcal{I}} \times \ldots \times \Delta^{\mathcal{I}}$ such that when we substitute the variables $\vec{x}$ with the objects $\vec{c}$, the formula $\exists \vec{y}.conj(\vec{x}, \vec{y})$ evaluates to true in $\mathcal{I}$. Analogously, a union of conjunctive query $q = \{ \vec{x} \mid \bigvee_{i=1,\ldots,n} \exists \vec{y}_i.conj_i(\vec{x}, \vec{y}_i) \}$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{c} \in \Delta^{\mathcal{I}} \times \ldots \times \Delta^{\mathcal{I}}$ such that when we substitute the variables $\vec{x}$ with the objects $\vec{c}$, the formula $\bigvee_{i=1,\ldots,n} \exists \vec{y}_i.conj_i(\vec{x}, \vec{y}_i)$ evaluates to true in $\mathcal{I}$.

As for boolean queries, given a boolean CQ $q = \{ \mid \exists \vec{y}.conj(\vec{y}) \}$ over a *DL-Lite* KB $\mathcal{K}$, and an interpretation $\mathcal{I}$ for $\mathcal{K}$, $q^{\mathcal{I}}$ consists of the only *empty tuple*, i.e., the tuple of arity 0, in the case in which the sentence $\exists \vec{y}.conj(\vec{y})$ is true in $\mathcal{I}$, whereas $q^{\mathcal{I}}$ is empty if such a sentence is false in $\mathcal{I}$. Analogously for boolean UCQs.

Notice that very recently, under WPs 4 and 6 of the TONES project, some extensions of the DL presented above have been considered, which have the same characteristics with respect computational complexity and possibility of solving reasoning by means of first-order query evaluation over a relational database instance [CDGL$^+$]. In fact, the term *DL-Lite* refers to an entire family of DLs, and the DL considered here is only one of such DLs (also called *DL-Lite*$_{\mathcal{F}}$, for its capability of specifying functionality restrictions on roles). We point out, however, that all results presented in the following apply, provided minor adaptations, to all DLs of the *DL-Lite* family.

## 14.3 Query Satisfiability

As already said, given a query $q$ specified over an ontology $\mathcal{O}$, *query satisfiability* is the problem of verifying whether there exists a model of $\mathcal{O}$ in which the query is evaluated to a non-empty set. For a *DL-Lite* KB and a conjunctive query, the problem can be formalized as follows. Given a *DL-Lite* TBox $\mathcal{T}$ and a conjunctive query $q$ over $\mathcal{T}$, we say that $q$ is satisfiable wrt $\mathcal{T}$ if there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $q^{\mathcal{M}} \neq \emptyset$.

In what follows we provide a technique for conjunctive query satisfiability over *DL-Lite* KBs. The technique is based on a reduction of the above problem to the problem knowledge base satisfiability in *DL-Lite*. We recall that, given a *DL-Lite* KB $\mathcal{K}$, the problem of KB satisfiability for $\mathcal{K}$ is the problem of checking whether there exists an interpretation $\mathcal{I}$ for $\mathcal{K}$ such that $\mathcal{I} \models \mathcal{K}$. Such a problem is under investigation within the TONES project under the Work Package 4 (WP4 – "Ontology access, processing, and usage") and Work Package 6 ( WP6 – "Ontology interoperation"), within which a technique for checking satisfiability of *DL-Lite* KBs has provided. Such a technique has been recently presented in [CDGL$^+$]. We now first briefly recall this technique and then provide our algorithm for query satisfiability.

### 14.3.1 An algorithm for KB satisfiability

We first introduce two preliminary definitions.

**Definition 14.1** Given an ABox $\mathcal{A}$ we denote by $db(\mathcal{A}) = (\Delta^{db(\mathcal{A})}, \cdot^{db(\mathcal{A})})$ the interpretation defined as follows:

$\Delta^{db(\mathcal{A})}$ is a non-empty set which contains all constants occurring in $\mathcal{A}$,

$a^{db(\mathcal{A})} = a$, for each constant $a$,

$A^{db(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, for each atomic concept $A$, and

$P^{db(\mathcal{A})} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$, for each atomic role $P$.

Roughly speaking, $db(\mathcal{A})$ is a minimal interpretation of the ABox $\mathcal{A}$, in which we interpret every constants occurring in $\mathcal{A}$ by itself.

**Definition 14.2** Let $\mathcal{T}$ be a *DL-Lite* TBox. We call *NI-closure of* $\mathcal{T}$, denoted by $cln(\mathcal{T})$, the TBox obtained inductively as follows:

1. all negative inclusion assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;
2. all functionality assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;
3. if $B_1 \sqsubseteq B_2$ is in $\mathcal{T}$ and $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ is in $cln(\mathcal{T})$, then also $B_1 \sqsubseteq \neg B_3$ is in $cln(\mathcal{T})$;
4. if one of the assertions $\exists r \sqsubseteq \neg\exists r$, or $\exists r^- \sqsubseteq \neg\exists r^-$ is in $cln(\mathcal{T})$, then both such assertions are in $cln(\mathcal{T})$;

In other words, $cln(\mathcal{T})$ is a special TBox that does not contain PIs and is obtained by closing the NIs in $\mathcal{T}$ with respect to the PIs in $\mathcal{T}$ (notice, however, that not all NIs logically implied $\mathcal{T}$ are inserted in $cln(\mathcal{T})$, but only those that are useful for our aims).

The following notable theorem tells us how to check satisfiability of a *DL-Lite* KB.

**Theorem 14.3** *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a DL-Lite KB. Then, $\mathcal{K}$ is satisfiable if and only if $db(\mathcal{A})$ is a model of the DL-Lite KB $(cln(\mathcal{T}), \mathcal{A})$.*

At this point, it is not difficult to show that verifying if $db(\mathcal{A})$ is a model of $(cln(\mathcal{T}), \mathcal{A})$ can be done by simply evaluating a suitable boolean conjunctive queries (with inequalities) over $db(\mathcal{A})$ itself. In particular we define a translation function $\delta$ from assertions in $cln(\mathcal{T})$ to boolean conjunctive queries with inequalities, as follows

$$\begin{array}{lcl} \delta((\mathsf{funct}\ r)) & = & \exists x, y, z.r(x, y) \wedge r(x, z) \wedge y \neq z \\ \delta((\mathsf{funct}\ r^-)) & = & \exists x, y, z.r(x, y) \wedge r(z, y) \wedge x \neq z \\ \delta(B_1 \sqsubseteq \neg B_2) & = & \exists x.\gamma_1(x) \wedge \gamma_2(x) \end{array}$$

where in the last equation $\gamma_i(x) = A_i(x)$ if $B_i = A_i$, $\gamma_i(x) = \exists y_i.r_i(x, y_i)$ if $B_i = \exists r_i$, and $\gamma_i(x) = \exists y_i.r_i(y_i, x)$ if $B_i = \exists r_i^-$.

The algorithm Consistent, described in Figure 14, takes as input a *DL-Lite* KB, computes $db(\mathcal{A})$ and $cln(\mathcal{T})$, and evaluates over $db(\mathcal{A})$ the boolean union of conjunctive queries obtained by taking the union of all boolean conjunctive queries returned by the application of the above function $\delta$ to every assertion in $cln(\mathcal{T})$.

In the algorithm, the symbol $\bot$ indicates a predicate whose evaluation is *false* in every interpretation. Therefore, in case in which neither functionality assertions nor negative inclusion assertions occur in $\mathcal{K}$, $q_{unsat}^{db(\mathcal{A})} = \bot^{db(\mathcal{A})}$, and therefore Consistent$(\mathcal{K})$ returns *true*.

The following theorem establishes soundness and completeness of the above technique.

**Algorithm** Consistent($\mathcal{K}$)
**Input:** *DL-Lite* knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$
**Output:** *true* if $\mathcal{K}$ is satisfiable, *false* otherwise
**begin**
  $q_{unsat} = \bot$;
  **for each** $\alpha \in cln(\mathcal{T})$ **do**
    $q_{unsat} = q_{unsat} \vee \delta(\alpha)$;
  **if** $q_{unsat}^{db(\mathcal{A})} = \emptyset$ **return** *true*;
  **else return** *false*;
**end**

Figure 14: The algorithm Consistent

**Lemma 14.4** *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a DL-Lite KB. Then, the algorithm Consistent($\mathcal{K}$) terminates, and $\mathcal{K}$ is satisfiable if and only if Consistent($\mathcal{K}$) = true.*

As for computational complexity, the following result holds

**Lemma 14.5** *In DL-Lite knowledge base satisfiability is* PTIME *in the size of the whole knowledge base (combined complexity).*

### 14.3.2 An algorithm for query satisfiability

In Figure 15 we present the algorithm QueryConsistent that we define for verifying query satisfiability. In the algorithm we make use of three functions.

- The function $\tau$, which takes as input a CQ and transforms it in a CQ without atoms with predicate symbol of the form $\exists r$ or $\exists r^-$. More precisely, $\tau$ takes as input a CQ $q = \{ \vec{x} \mid \exists \vec{y}.conj(\vec{x}, \vec{y}) \}$ and returns a CQ $q' = \{ \vec{x} \mid \exists \vec{y'}.conj'(\vec{x}, \vec{y'}) \}$, where $conj'(\vec{x}, \vec{y'})$ is obtained from $conj(\vec{x}, \vec{y})$ by replacing each atom of the form $\exists r(z)$ (resp. $\exists r^-(z)$), where $z$ is a constant or a variable in $\vec{x}$ or $\vec{y}$, with an atom of the form $r(z, z')$ (resp. $r(z', z)$), where $z'$ is a new variable not occurring elsewhere in $q'$, and $\vec{y'} = \vec{y}, \vec{z}$, where $\vec{z}$ is the sequence of the new variables that have bee introduced in the transformation of the query.

- The function *freeze*, takes as input a conjunctive query $q$, and returns a set of *DL-Lite* membership assertions obtained by "freezing" the body of $q$, i.e., considering its atoms as membership assertions in an ABox. More precisely, given a CQ $q = \{ \vec{x} \mid \exists \vec{y}.conj(\vec{x}, \vec{y}) \}$, we have that

$$freeze(q) = \{A(z) \mid A(z) \in conj(\vec{x}, \vec{y})\} \cup \{r(z_1, z_2) \mid r(z_1, z_2) \in conj(\vec{x}, \vec{y})\}$$

  where $z$, $z_1$, $z_2$ are constants or variables in $\vec{x}$ or $\vec{y}$. Notice that the set of constants in the ABox returned by the function *freeze* is the union of the set of constants occurring in the query $q$ and the set of variables in the query $q$ (therefore, variables in the query are now considered ABox constants).

**Algorithm** QueryConsistent$(q, \mathcal{T})$
**Input:** Conjunctive Query $q$; *DL-Lite* TBox $\mathcal{T}$
**Output:** *true* if $q$ is satisfiable wrt $\mathcal{T}$, *false* otherwise
**begin**
  **let** $\Gamma_V$ be the set of variables occurring in $q$;
  $\mathcal{A}_q = \mathit{unify}(\mathit{freeze}(\tau(q)), \mathcal{T}, \Gamma_V)$;
  $\mathcal{K}_q = (\mathcal{T}, \mathcal{A}_q)$;
  **return** Consistent$(\mathcal{K}_q)$;
**end**

Figure 15: The algorithm QueryConsistent

- the function *unify*, which takes as input a *DL-Lite* ABox $\mathcal{A}$, a *DL-Lite* TBox $\mathcal{T}$ and a set of constants symbol $\Gamma$, and returns a *DL-Lite* ABox $\mathcal{A}'$ obtained from $\mathcal{A}$ by unifying all constants from $\Gamma$ occurring in $\mathcal{A}$ according to functionality assertions contained in $\mathcal{T}$. More precisely, $\mathcal{A}' = \mathit{unify}(\mathcal{A}, \mathcal{T}, \Gamma)$, is obtained starting from $\mathcal{A}$ by repeatedly applying the following rule that unifies constants of $\Gamma$:

  FUNCT-RULE: if the functionality assertion (funct $r$) (resp. (funct $r^-$)) is in $\mathcal{T}$ and there are two membership assertions $r(z, z'), r(z, z'')$ (resp. $r(z', z), r(z'', z)$) in $\mathcal{A}'$ such that either $z'$ or $z''$ occur in $\Gamma$, then $(a)$ if $z''$ occurs in $\Gamma$, replace each occurrence of $z''$ in $\mathcal{A}'$ with $z'$, or $(b)$ if $z''$ does not occur in $\Gamma$, replace each occurrence of $z'$ in $\mathcal{A}'$ with $z''$.

Notice that the use of the function $\tau$ is necessary due to the presence in the query $q$ of atoms of the form $\exists r(z)$ and $\exists r^-(z)$ that cannot be directly freezed by the function *freeze*. Furthermore, the use of the function *unify* is needed due to the presence of functionality assertions in $\mathcal{T}$. Indeed, the basic idea of the algorithm is to construct an ABox which suitably represents the query[21], and then verify that the knowledge base obtained by uniting such an ABox with the TBox $\mathcal{T}$ is satisfiable. However, unsatisfiability of such a knowledge base might arise since two membership assertions of the form $r(z, z')$ and $r(z, z'')$, where $z'$ and $z''$ do not both belong to $\Gamma_V$, violate a functionality assertion of the form (funct $r$), but this is not a real violation, since $z'$ and $z''$ can be unified. In other words, due the presence of functionality assertions, constants in the ABox coming from the "freezing" of variables could be made equal.

The following theorem establishes that the algorithm QueryConsistent is sound and complete.

**Theorem 14.6** *Let $\mathcal{T}$ be a DL-Lite TBox, and let $q$ a conjunctive query over $\mathcal{T}$. Then, the algorithm* QueryConsistent$(q, \mathcal{T})$ *terminates, and $q$ is satisfiable wrt $\mathcal{T}$ if and only if* QueryConsistent$(q, \mathcal{T}) = \mathit{true}$.

We are finally able to provide computational complexity of query satisfiability (only membership).

[21]The technique used for achieving this aim is called freezing of the query and is typically used in the reduction of query containment to query answering in the database theory [AHV95]

**Algorithm** QueryDisjointness$(q_1, q_2, \mathcal{T})$
**Input:** CQ $q_1 = \{ \vec{x} \mid \exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1}) \}$, CQ $q_2 = \{ \vec{x} \mid \exists \vec{y}.conj_2(\vec{x}, \vec{y_2}) \}$
*DL-Lite* TBox $\mathcal{T}$
**Output:** *true* if $q_1$ and $q_2$ are disjoint wrt $\mathcal{T}$, *false* otherwise
**begin**
$q = \{ \vec{x} \mid \exists \vec{y_1}, \vec{y_2}.conj_2(\vec{x}, \vec{y_2}) \wedge conj_2(\vec{x}, \vec{y_2}) \}$;
**return not** QueryConsistent$(q, \mathcal{T})$;
**end**

Figure 16: The algorithm QueryDisjointness

**Theorem 14.7** *Let $\mathcal{T}$ be a DL-Lite TBox, and let $q$ a conjunctive query over $\mathcal{T}$. Then, establishing if $q$ is satisfiable wrt $\mathcal{T}$ is* PTIME *in the size of the query and the TBox (combined complexity).*

## 14.4 Query Disjointness

As already said, given two queries $q_1$ and $q_2$ specified over an ontology $\mathcal{O}$, *query disjointness* is the problem of verifying whether, for every model of $\mathcal{O}$, the answer sets of $q_1$ and $q_2$ are always mutually exclusive. For *DL-Lite* KBs and conjunctive queries, the problem can be formalized as follows. Given a *DL-Lite* TBox $\mathcal{T}$ and two conjunctive queries $q_1$ and $q2$ of the same arity over $\mathcal{T}$, we say that $q_1$ and $q_2$ are disjoint wrt $\mathcal{T}$ if for every model $\mathcal{M}$ of $\mathcal{T}$ we have that $q_1^{\mathcal{M}} \cap q_2^{\mathcal{M}} = \emptyset$.

In Figure 16 we present the algorithm QueryDisjointness that we define for checking query disjointness. In the algorithm we construct a new conjunctive query $q$ that is obtained by the conjunction of the the two conjunctive queries in input. Then, we can conclude that the two queries are disjoint if the new query $q$ is unsatisfiable (and vice-versa), and for verifying this we make use of the algorithm for query satisfiability defined in the above subsection. The following theorem shows that QueryDisjointness is sound and complete.

**Theorem 14.8** *Let $\mathcal{T}$ be a DL-Lite TBox, and let $q_1$ and $q_2$ two conjunctive queries over $\mathcal{T}$. Then, the algorithm* QueryDisjointness$(q_1, q_2, \mathcal{T})$ *terminates, and $q_1$ and $q_2$ are disjoint wrt $\mathcal{T}$ if and only if* QueryDisjointness$(q_1, q_2, \mathcal{T}) = true$.

We are now able to provide computational complexity of conjunctive query disjointness in *DL-Lite* (only membership).

**Theorem 14.9** *Let $\mathcal{T}$ be a DL-Lite TBox, and let $q_1$ and $q_2$ be two conjunctive queries over $\mathcal{T}$. Then, establishing if $q_1$ and $q_2$ are disjoint wrt $\mathcal{T}$ is* PTIME *in the size of the queries and the TBox (combined complexity).*

## 14.5 Query Containment

As already said, given two queries $q_1$ and $q_2$ specified over an ontology $\mathcal{O}$, *query containment of $q_1$ in $q_2$* is the problem of verifying whether, for every model of $\mathcal{O}$, the answer

**Algorithm** QueryContainment$(q_1, q_2, \mathcal{T})$
**Input:** CQs $q_1$ and $q_2$, *DL-Lite* TBox $\mathcal{T}$
**Output:** *true* if $q_1$ is contained in $q_2$ wrt $\mathcal{T}$, *false* otherwise
**begin**
  **if not** QueryConsistent$(q_1, \mathcal{T})$
    **return** *true*;
  **else if not** QueryConsistent$(q_2, \mathcal{T})$
      **return** *false*;
  **else begin**
    **let** $\Gamma_V$ be the set of variables occurring in $q_1$;
    $\mathcal{A}_{q_1} = \mathit{unify}(\mathit{freeze}(\tau(q_1)), \mathcal{T}, \Gamma_V)$;
    $PR_2 = \mathit{perfectRef}(q_2, \mathcal{T})$;
    **if** $PR_2^{db(\mathcal{A}_{q_1})} = \emptyset$
      **return** *false*;
    **else return** *true*;
  **end**
**end**

Figure 17: The algorithm QueryContainment

set of $q_1$ is contained in the answer set of $q_2$. For *DL-Lite* KBs and conjunctive queries, the problem can be formalized as follows. Given a *DL-Lite* TBox $\mathcal{T}$ and two conjunctive queries $q_1$ and $q_2$ of the same arity over $\mathcal{K}$, we say that $q_1$ *is contained in* $q_2$ *wrt* $\mathcal{T}$, denoted $q_1 \subseteq_{\mathcal{T}} q_2$ if for every model $\mathcal{M}$ of $\mathcal{T}$ we have that $q_1^{\mathcal{M}} \subseteq q_2^{\mathcal{M}}$.

In the following, we present an algorithm for query containment which is based on the reduction of this problem to a problem of query answering over a *DL-Lite* KB.

We recall that, given a query $q$ (either a conjunctive query or a union of conjunctive queries) and a *DL-Lite* KB $\mathcal{K}$, the *answer to* $q$ *over* $\mathcal{K}$ is the set $\mathit{ans}(q, \mathcal{K})$ of tuples $\vec{a}$ of constants appearing in $\mathcal{K}$ such that $\vec{a}^{\mathcal{M}} \in q^{\mathcal{M}}$, for every model $\mathcal{M}$ of $\mathcal{K}$. Notice that by definition $\mathit{ans}(q, \mathcal{K})$ is finite since $\mathcal{K}$ is finite, and hence the number of constants appearing in $\mathcal{K}$ is finite. Notice also that the tuple $\vec{a}$ can be the empty tuple in the case in which $q$ is a boolean conjunctive query. More precisely, in this case the set $\mathit{ans}(q, \mathcal{K})$ consists of the only empty tuple if and only if the formula $q$ is true in every model of $\mathcal{K}$.

The problem of (conjunctive) query answering in *DL-Lite* is currently under investigation within the TONES project under the Work Package 4 (WP4 – "Ontology access, processing, and usage") and Work Package 6 ( WP6 – "Ontology interoperation"). Within these WPs a technique for answering union of conjunctive queries over *DL-Lite* KBs has been produced and recently presented in [CDGL$^+$] (a preliminary version of it has been published in [CDGL$^+$05a]).

In a nutshell, our query answering method strongly separates the intensional and the extensional level of the *DL-Lite* KB: the query is first processed and reformulated based on the TBox axioms; then, the TBox is discarded and the reformulated query is evaluated over the ABox, as if the ABox were a simple relational database (cf. Definition 14.1). More precisely, given a (union of) conjunctive query $q$ given a UCQ $q$ over a *DL-Lite*

$\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we first compute the so-called *perfect reformulation of q wrt* $\mathcal{T}$, i.e., a query $q_r$ such that, for each ABox $\mathcal{A}'$, $q_r^{db(\mathcal{A}')} = ans(q, (\mathcal{T}, \mathcal{A}')$. As shown in [CDGL$^+$], in the above setting $q_r$ is a union of conjunctive queries. Therefore, when then evaluating $q_r$ over $db(\mathcal{A})$ in order to get the answer to $q$ over $\mathcal{K}$, we essentially reduce query answering to the evaluation of a union of conjunctive queries over a database instance. In order to compute the perfect reformulation, we defined an algorithm, called *perfectRef*, which takes as input a UCQ $q$ and a *DL-Lite* TBox $\mathcal{T}$, and produces a union of conjunctive queries $PR$ over $\mathcal{T}$ that is the perfect reformulation of $q$ of $q$ under $\mathcal{T}$. Roughly speaking, in *perfectRef*, PIs in $\mathcal{T}$ are used as rewriting rules, iteratively applied from right to left to atoms occurring in the query body, which allow one to compile away in the reformulation the intensional knowledge (represented by $\mathcal{T}$) that is relevant for answering $q$. We do not give here the exact definition of the algorithm *perfectRef*, but refer the reader to [CDGL$^+$05a, CDGL$^+$] for such details and for the formal proof that *perfectRef* is sound and complete with respect to the problem of computing perfect rewriting.

Let us now turn our attention to our algorithm for query containment. In Figure 17 we present the algorithm QueryContainment. For the sake of simplicity, QueryContainment takes as input only boolean conjunctive queries. However, the case of non-boolean conjunctive queries can be dealt with in an analogous way, provided minor modification to the algorithm. In the algorithm we make use of the functions $\tau$, *freeze*, *unify* introduced in the subsection on query satisfiability, and of the function *perfectRef*, discussed before. Furthermore, $db(\mathcal{A}_{q_1})$ represents the special interpretation of Definition 14.1 for the ABox $\mathcal{A}_{q_1}$, obtained by the freezing of the query $q_1$.

Roughly speaking, in the case in which both $q_1$ and $q_2$ are consistent, the algorithm QueryContainment first "freezes" the body of the query $q_1$, thus obtaining $\mathcal{A}_{q_1}$ (such a process has been precisely described in the subsection on query satisfiability), and then compute the answer to $q_2$ over $(\mathcal{T}, \mathcal{A}_{q_1})$, exploiting the algorithm *perfectRef* and the properties of the perfect reformulation. In the case in which $q_1$ is unsatisfiable, the containment of $q_1$ in $q_2$ is trivially true, since evaluation of $q_1$ is empty in every interpretation. Similarly, if $q_2$ is unsatisfiable and $q_1$ is satisfiable, the containment of $q_1$ in $q_2$ is false, since evaluation of $q_2$ is empty in every interpretation, but $q_1$ is evaluated to a non-empty set in at least one interpretation.

The following theorem shows soundness and correctness of the algorithm QueryContainment.

**Theorem 14.10** *Let $\mathcal{T}$ be a DL-Lite TBox, and let $q_1$ and $q_2$ two conjunctive queries over $\mathcal{T}$. Then, the algorithm* QueryContainment$(q_1, q_2, \mathcal{T})$ *terminates, and $q_1 \subseteq_{\mathcal{T}} q_2$ if and only if* QueryContainment$(q_1, q_2, \mathcal{T}) = true$.

We finally provide computational complexity of conjunctive query containment in *DL-Lite*.

**Theorem 14.11** *Let $\mathcal{T}$ be a DL-Lite TBox, and let $q_1$ and $q_2$ two conjunctive queries over $\mathcal{T}$. Then, establishing if $q_1 \subseteq_{\mathcal{T}} q_2$ is NP-complete in the size of the queries and the TBox (combined complexity).*

We point out that, notably, the complexity of the above problem in *DL-Lite* is exactly the same of the complexity of the containment of conjunctive queries over relational databases [CM77].

## 14.6 Related Work

Among the reasoning services considered in this Section, the most important and the one that has received more attention is the problem of query containment. Many papers point out that checking containment is a relevant task in several contexts, including information integration [Ull97], query optimization [AHV95, ASU79a], (materialized) view maintenance [GM95], data warehousing [Wid95], constraint checking [GSUW94], and semantic caching [APTP03].

Query containment under integrity constraints, is the problem of checking whether containment between two queries holds for every interpretation satisfying a given set of constraints (which may be classical relational database constraints in a database setting, or DL constraints in a context in which ontologies come into the scene).

This problem arises in those situation where one wants to check query containment relatively to a data schema specified with a rich data definition language. For example, in the case of information integration, queries are often to be compared relatively to (inter-schema) constraints, which are used to declaratively specify the "glue" between two source schemas, and between one source schema and the global schema [CDGL$^+$98b, Hul97, Ull97, CL93, LSK95, Len02, Hal01].

The complexity of query containment in the absence of constraints has been studied in various settings. In [CM77], NP-completeness has been established for conjunctive queries, and in [CR97] a multi-parameter analysis has been performed for the same case, showing that the intractability is due to certain types of cycles in the queries. In [Klu88, vdM98], $\Pi_2^p$-completeness of containment of conjunctive queries with inequalities was proved, and in [SY80] the case of queries with the union and difference operators was studied. For various classes of Datalog queries with inequalities, decidability and undecidability results were presented in [CV92, vdM98, Bon04, CDGV03], respectively.

Query containment under constraints has also been the subject of several investigations. For example, decidability of conjunctive query containment was investigated in [ASU79b] under functional and multi-valued dependencies, in [JK84] under functional and inclusion dependencies, in [Cha92, LR96, LS97] under constraints representing *is-a* hierarchies and complex objects, and in [DS96] in the case of constraints represented as Datalog programs. Undecidability is proved in [CR03] for recursive queries under inclusion dependencies. Several results on containment of XML queries under constraints expressed as DTDs are reported in [NS03, Woo03]. Finally, conjunctive query containment under description logics constraints has been studied in [CDGL98a], where algorithms for decidable cases are given, and undecidability of containment of conjunctive queries with inequalities in the right-hand side query is proved.

# 15 Ontology specification in natural language

## 15.1 Introduction

In this section we address the issue of providing a natural language interface to ontology design tools. The aim of such an interface is to allow unskilled users to state in natural language the assertions, both at the intensional and at the extensional level, that are to become part of an ontology. The problem we are considering here goes back to early work in the Seventies and Eighties to build Natural Language Interfaces (NLIs) to Databases, and that has turned out to some level of disappointment towards the Nineties [ART95]. Due to the recent developments both in Knowledge Representation and in Natural Language Technologies, and the widespread actual and foreseen use of ontologies, e.g., in the Semantic Web [BKGK05, ST04a], a similar question is calling for attention again. Specifically, due to the restricted expressive power of current ontology languages, NLIs are required where only a suitable fragment of natural language (a *controlled natural language*) is used [Huj98, Kit03, Sow04].

For the purpose of querying data sources through ontologies, systems have been proposed that guide the user to formulate her question via an ontology that incrementally shows the possible concepts on which the remaining part of the question could be about [DFT04, DF06]. Others guide the user via an incremental parser [BKKK06, ST04b]. Both approaches aim at allowing the user to build only those questions that the system can handle (and that form a controlled language). Differently from those approaches, which concentrate on queries, here we aim at providing a natural language interface for the specification of an ontology. We concentrate on ontologies expressed in DLs that belong to the *DL-Lite* family [CDGL$^+$05a, CDGL$^+$06]. Such DLs are designed and specifically optimized for the purpose of providing efficient access to large data repositories [CDGL$^+$05b].

Specifically, we address the question of *which* should be the natural language fragment to be used for such a purpose, and *how* we can define it. To this end we consider of particular value the studies carried out by Ian Pratt [PHT06] who is investigating the satisfiability of sets of sentences in fragments of natural language and its computational complexity. Our proposal is similar in spirit to Pratt's approach, in that we aim at identifying natural language fragments with a desirable computational complexity; we use as controlled language for accessing ontologies fragments for which the meaning representation of a sentence is expressible as a *DL-Lite* (TBox or ABox) assertion, and hence can be efficiently reasoned upon. Instead, we resort to *Categorial Grammar* (CG) as the system to answer the *how*, viz. aiming to capture those syntactic structures corresponding to all and only the meaning representations allowed in the chosen variant of *DL-Lite*.

## 15.2 The framework

Our aim is to build a natural language interface that helps users to specify an ontology expressed in a DL, i.e., to enter facts in the ABox and universal statements in the TBox. To this end, we follow the Controlled Language approach to the problem. Since we are interested in performing these tasks over large data, efficiency in managing such data through the ontology is of primary importance. For this reason, we focus our attention

on *DL-Lite* [CDGL$^+$05a], a family of DLs specifically tailored to manage large amounts of data efficiently. More precisely, answering unions of conjunctive queries[22] over a *DL-Lite* ontology is polynomial in the size of the TBox, and LogSpace in the size of the ABox, and, most importantly, it can be done by relying on a commercial RDBMS system for storing the ABox and managing the access to its data. Moreover, the DLs in the *DL-Lite* family are essentially the maximal DLs that exhibit such nice computational properties [CDGL$^+$06]. Nevertheless, they have the ability to capture the essential features of the most commonly used formalisms for conceptual modeling, such as UML class diagrams and entity-relationship schemas [CDGL$^+$05a].

We want to capture the fragment of English that consists of all and only those sentences whose meaning representation belongs to *DL-Lite*. This fragment is rather restricted and might turn out to be too constrained for a user. Therefore, we plan to study how natural language structures that outscope the defined fragments can be rephrased so to maintain their meaning at least partially while satisfying the constraints imposed by the defined grammar, and therefore be reduced to eligible constructs.

The idea of studying a Controlled Language, though brought up for different motivations, is not too far away from Montague's proposal to restrict attention to fragments of natural languages [Tho74]. His program was based on the thesis that the relation between syntax and semantics in a natural language such as English could be viewed as not essentially different from the relation between syntax and semantics in a formal language such as the language of FOL. The main components of his framework are: (i) the principle of compositionality (i.e., the meaning of the whole is built out of the meaning of its parts) together with the idea that the construction of meaning is guided by the syntactic structure; (ii) the view of words (and phrases) as complete (e.g., noun phrases) and incomplete (e.g., verbs) expressions, and consequently their representation as functions by means of lambda terms and the assignment of categorial grammar types as syntactic categories; and finally (iii) the model-theoretic interpretation of the obtained meaning representations based on standard FOL interpretation. Following Montague, we use a Categorial Grammar (CG) to capture the needed natural language fragment. Furthermore, we exploit the syntax-semantics interface provided by the Curry-Howard Correspondence between the Lambek Calculus (the logic version of CG) and the lambda calculus to obtain *DL-Lite* meaning representation compositionally while parsing [Ben87, Moo97].

## 15.3 *DL-Lite* and Natural Language

### 15.3.1 Introduction to *DL-Lite*

In this work, we consider DLs belonging to the *DL-Lite* family [CDGL$^+$05a, CDGL$^+$06], and specifically, we consider variants of *DL-Lite* in which the TBox is constituted by a set of *inclusion assertions* of the form

$$Cl \sqsubseteq Cr$$

[22]Such queries correspond to unions of select-project-join SQL queries, which constitute the vast majority of queries posed to RDBMS systems.

where $Cl$ and $Cr$ denote concepts that may occur respectively on the left and right-hand side of inclusion assertions. The form of such concepts depends on the specific variant of *DL-Lite*. Here, we consider two variants, called *DL-Lite*$_{core}$ and *DL-Lite*$_{\mathcal{R},\sqcap}$, and defined below. In fact, *DL-Lite*$_{core}$ represents a core part shared by all logics of the *DL-Lite* family.

**Definition 15.1** [*DL-Lite*$_{core}$ and *DL-Lite*$_{\mathcal{R},\sqcap}$ ] In *DL-Lite*$_{core}$, $Cl$ and $Cr$ are defined as follows:[23]

$$Cl \longrightarrow A \mid \exists r \qquad\qquad Cr \longrightarrow A \mid \neg A \mid \exists r \mid \neg\exists r$$

where $A$ denotes an atomic concept, and $r$ denotes an atomic role.

In *DL-Lite*$_{\mathcal{R},\sqcap}$, in addition to the clauses of *DL-Lite*$_{core}$, we have also:

$$Cl \longrightarrow Cl_1 \sqcap Cl_2 \qquad\qquad Cr \longrightarrow \exists r.A$$

*DL-Lite* can be seen as a restricted form of the DLs introduced in Section 2, and its semantics can be defined as for those DLs.

Finally, in *DL-Lite*, an ABox is constituted by a set of assertions on *individuals*, of the form $A(c)$ or $r(a, b)$, where $A$ and $r$ denote respectively an atomic concept and role, and $a$, and $b$ denote constants. As in FOL, each constant is interpreted as an element of the interpretation domain, and the above ABox assertions correspond to the analogous FOL facts.

We are interested in studying the linguistic structures that correspond to such constructs. In the following we look at their straightforward translations into natural language.

### 15.3.2 Fragment of Natural Language for *DL-Lite*

The constraints expressed in the TBox are universals. They are of the form $Cl \sqsubseteq Cr$ that translates into FOL as $\forall x.Cl(x) \rightarrow Cr(x)$ and in natural language as

(a) [Every $\underbrace{\texttt{NOUN}}_{Cl}$] $\underbrace{\texttt{VERB\_PHRASE}}_{Cr}$

(b) [[Everyone $\underbrace{[\text{who } \texttt{VERB\_PHRASE}]}_{Cl}$] $\underbrace{\texttt{VERB\_PHRASE}}_{Cr}$]

Hence, the determiner "every" and the quantifier "everyone" play a crucial role in determining the linguistic structures that belong to the natural language fragment corresponding to a *DL-Lite* TBox. In the following, we zoom into the NOUN and VERB_PHRASE constituents. In other words, we spell out how the $Cl$ and $Cr$ of *DL-Lite* can be expressed in English.

First of all, notice that since an atomic concept $A$ is a unary predicate, it can be either a noun "student", e.g., (2) or an intransitive verb phrase "left" (1); and its negation, $\neg A$, can be either "is not a boy" (3), or "does not leave" (4).

[23] We have omitted *inverse* roles from the DLs to simplify the presentation of the main idea we are investigating.

The introduction of the $\exists r$ in the $Cl$ part can be performed by means of the quantifier "everyone" followed by the relative pronoun "who"(5 and 6) (or by the conjunction that would correspond to the use of $\sqcap$ on the $Cl$ part allowed in *DL-Lite*$_{\mathcal{R},\sqcap}$ fragment, see (13) below).

(1) Every student left $[Student \sqsubseteq Left]$

(2) Every student is a boy $[Student \sqsubseteq Boy]$

(3) Every student is not a boy $[Student \sqsubseteq \neg Boy]$

(4) Every student does not leave $[Student \sqsubseteq \neg Leave]$

(5) Everyone who eats left $[\exists Eats \sqsubseteq Left]$

(6) Everyone who knows something left $[\exists Know \sqsubseteq Left]$

On the other hand, the introduction of $\exists R$ on the $Cr$ part corresponds to the use of a transitive verb followed by an existential quantifier, "something" (7), and its negation to the use of "does not" to negate such construction (8).

(7) Every student knows something $[Student \sqsubseteq \exists Know]$

(8) Every student does not know something $[Student \sqsubseteq \neg\exists Know]$

Note, that as the *DL-Lite* clause show the only reading of the ambiguous sentence in (8) is the one with *every* having wide scope and *something* be in the scope of *not*[24].

When we move to *DL-Lite*$_{\mathcal{R},\sqcap}$, the addition of the conjunction in the $Cl$ part corresponds to the use of adjective (9), or relative clauses modifying the noun quantified by "every" (10-12), or the "and" coordinating two verb phrases (13).

(9) Every nice student left.
$\forall x.(\texttt{student}(x) \wedge \texttt{nice}(x)) \rightarrow \texttt{left}(x)$ $[Student \sqcap Nice \sqsubseteq Left]$

(10) Every student who studies left.
$\forall x.(\texttt{student}(x) \wedge \texttt{study}(x)) \rightarrow \texttt{left}(x)$ $[Student \sqcap \exists Study \sqsubseteq Left]$

(11) Every student who is a boy left.
$\forall x.(\texttt{student}(x) \wedge \texttt{Boy}(x)) \rightarrow \texttt{left}(x)$ $[Student \sqcap Boy \sqsubseteq Left]$

(12) Every student who eats something left.
$\forall x.(\texttt{student}(x) \wedge \exists y.\texttt{eats}(x,y)) \rightarrow \texttt{left}(x)$ $[Student \sqcap \exists Eats \sqsubseteq Left]$

(13) Everyone who drinks something and eats something left.
$\forall x.(\exists y.\texttt{drink}(x,y) \wedge \exists z.\texttt{eats}(x,z)) \rightarrow left(x)$ $[\exists Drinks \sqcap \exists Eats \sqsubseteq Left]$

[24]For ease of explanation we do not consider the distinction between *something* and the negative polarity item *anything*. This distinction could be incorporated into the fragment as studied in [Ber02].

Furthermore, the introduction of the qualified existential on the $Cr$ is performed by the determiner "a" (14).

(14) Every student knows a girl.

$\forall x.\texttt{student}(x) \rightarrow \exists y.\texttt{girl}(y) \wedge \texttt{know}(x, y)$ $[Student \sqsubseteq \exists Know.Girl]$

An important remark to emphasize is the presence of the relative pronoun in the above fragment of sentences. Pratt [PHT06] has shown how the uncontrolled use of such expression brings to `NP-complete` fragments when allowing the use only of the copula or even `EXPTIME-completeness` when adding transitive verbs. Below, we will show how relative pronouns can be used in a controlled grammar and preserve tractability of inferences.

We now turn to show how the Lambek Calculus can be used to capture the Natural Language fragment consisting *only* of linguistic structures listed above or a recursive extension of them. Notice that the latter can happen only by means of either conjunction (*and*) or an adjective.

## 15.4 CG and Natural Language for *DL-Lite*

### 15.4.1 Introduction to Categorial Grammar

As most of the linguistically motivated formal grammars currently in use, Categorial Grammar (CG) is a lexicalised grammar, i.e., the lexicon carries most of the information about how words can be assembled to form grammatical structures. The peculiarity of CG is that it captures the tight correspondence between syntax and semantics of natural language. In the logical version we employ, namely the (non associative) Lambek Calculus (NL) [Lam58, Moo97], this aspect is even stronger thanks to the Curry-Howard Correspondence that holds between the logical rules of NL and (a fragment of) typed lambda calculus [Ben87]. In this framework, syntactic categories are seen as *formulas* and their category forming operators as connectives, i.e., *logical constants*. As a result, the rules for category combination are the (very few) rules of inference for these connectives (function application and abstraction[25]). This aspect of the formalism significantly simplifies the implementation task, since one has to focus only on the construction of the lexicon and can rely on any existing parser for the Lambek Calculus.[26]

Information both about the syntactic structure where the word could occur and its meaning are stored in the lexicon.

**Definition 15.2** [Term Labelled Lexicon] Given a set $\Sigma$ of basic expressions of a natural language, a term labelled categorial lexicon is a relation,

$$\mathsf{LEX} \subseteq \Sigma \times (\mathsf{CAT} \times \mathsf{TERM}) \text{ such that if } (w, (A, \alpha)) \in \mathsf{LEX}, \text{ then } \alpha \in \mathsf{TERM}_{\mathtt{type}(A)}$$

where $\mathsf{TERM}$ is the set of all lambda terms and $\mathsf{TERM}_{\mathtt{type}(A)}$ denotes the set of lambda terms whose type is mapped to the category $A$. $\mathsf{CAT}$ is defined as follows

$$\mathsf{CAT} ::= \mathsf{ATOM} \mid \mathsf{CAT}\backslash\mathsf{CAT} \mid \mathsf{CAT}/\mathsf{CAT}$$

[25] In this paper we use the product free version of NL.

[26] The lexicon we present in this article has been tested using `Grail` [Moo98].

In the following, we will use the set of atoms $\mathsf{ATOM} = \{np, n, s\}$, and the function $\mathtt{type}$: $\mathsf{CAT} \rightarrow \mathsf{TYPE}$ mapping syntactic categories to semantic types given below, where the atomic types are $e$ (entity) and $t$ (truth values), and $(a, b)$ denotes the functional type $a \rightarrow b$ as always.

$$\begin{array}{ll} \mathtt{type}(np) = e; & \mathtt{type}(A/B) = (\mathtt{type}(B), \mathtt{type}(A)); \\ \mathtt{type}(s) = t; & \mathtt{type}(B\backslash A) = (\mathtt{type}(B), \mathtt{type}(A)); \\ \mathtt{type}(n) = (e, t). & \end{array}$$

This constraint on lexical entries enforces the requirement that if the expression $w$ is assigned a syntactic category $A$ and term $\alpha$, then the term $\alpha$ is of the appropriate type for the category $A$. We will assign lambda terms whose body is a FOL formula, $\lambda$-FOL.

We look at the determiner *every*, by means of example, since it has a crucial role in our grammar. The reader is referred to [KF85, Eij85] for an in depth explanation of this and similar expression.

**Example 15.3** [Determiner] The meaning of "every NOUN" (e.g., "every man") is the set of those properties that every NOUN (e.g., man) has

$$[\![\mathtt{every\ NOUN}]\!] = \{X | [\![\mathtt{NOUN}]\!] \subseteq X\}.$$

Therefore, in a functional perspective, it is seen as a two argument function that corresponds to the following syntactic category

$$(s/(np\backslash s))/n$$

where the $n$ is the first argument that must occur on the right of *every* and $np\backslash s$, i.e., a verb phrase, is its second argument to occur still on the right of *every* NOUN (viz. [[*every* NOUN] VP]). The typed lambda term corresponding to this syntactic category is: $\lambda X_{(e,t)}.\lambda Y_{(e,t)}.\forall x_e X(x) \rightarrow Y(x)$ that properly represents the set theoretical meaning given above. In the following, we won't use types on the lambda term unless necessary.

Our proposal for the definition of the proper fragment of natural language exploits this correspondence between syntactic categories and lambda terms.

Furthermore, it takes advantage of derivability relations among categories of the same semantic type carried out by unary operators decorating CAT [Ber02]. For reason of space, we won't go into the details of this part which would require the introduction of the multi modal extension of NL. It suffices to provide the intuitive idea behind the proposed solution: a function $A \rightarrow B$ can be applied to either an argument $A$ or to an argument $C$ such that $C$ derives $A$ ($C \Rightarrow A$). In our case, $\Rightarrow$ is the derivability relation of the logical grammar we use.

Finally, the "parsing as deduction" approach, gives us a mean to reduce the problem of identifying a proper set of linguistic structures to a problem of defining the allowed logical formulas. In other words, instead of looking at linguistic strings $w_1 \ldots w_n$, we can exploit the formally well defined structures corresponding to them. Parsing a string $w_1 \ldots w_n$ amounts to prove that $\Gamma \vdash B : \phi$, where $\Gamma$ consists of pairs of categories and terms as defined in the lexicon (viz. $(w_i, (A_i, \alpha_i))$), $A_1 : \alpha_1, \ldots A_n : \alpha_n$, to be proved to be of

category $B$. As by-product of this derivation one derives also the meaning representation of the structure assigned to the string, i.e., the lambda term $\phi$. For instance, parsing *Every nice student left* means to prove that the following is a theorem of NL:

$$(((s/(np\backslash s))/n : \texttt{every} \circ (n/n : \texttt{nice} \circ n : \texttt{student})) \circ np\backslash s : \texttt{left}) \vdash s : \phi$$

by using the proper lambda terms, through the proof of the above entailment, $\phi$ results to be $\forall x.(\texttt{nice}(x) \wedge \texttt{student}(x)) \rightarrow \texttt{left}(x)$.

### 15.4.2 *CG-Lite*

Our task is to find syntactic categories that lexically control the restrictions imposed by the *DL-Lite* constructs. We proceed step by step, by first looking at the requests of *DL-Lite*$_{core}$ and consider the two constraints regarding the use of negation:

1. negation of atomic concepts can occur in the $Cr$ part (i.e., $Cr \longrightarrow A \mid \neg A$) but not in the $Cl$ part (i.e., $Cl \longrightarrow A$);
2. an unqualified existential can occur both in $Cl$ and in $Cr$, but its negation can occur only in $Cr$ (i.e., $Cl \longrightarrow \exists r$, $Cr \longrightarrow \exists r \mid \neg\exists r$).

The $Cl$ and $Cr$ parts correspond to the restrictive scope (the `NOUN`), and nuclear scope (the `VP`) of *every*, respectively. We need to constrain the linguistic structures that occur in them. In particular, we need to block the occurrences of the negation in $Cl$ and express the fact that `NOT` cannot have its scope on any `VP` occurring in the restrictive scope of *every*. As emphasised in [Ber02], in CG scope is determined by the sentential categories $s$ of the complex formulas, and different scope distribution can be accounted for by differentiating the sentential levels of the scope constructors at work, and exploiting the derivability relations among categories.

We mark the structures that can occur in the two parts of the *DL-Lite* clauses and the negative and positive ones, by means of the four sentential levels $s_{cl}$, $s_{cr}$, $s_\neg$, and $s$, respectively, and establish the derivability relation below.[27] They state that a negated sentence can be in the $Cr$ construct ($s_\neg \Rightarrow s_{cr}$) while it cannot be in the $Cl$ part ($s_\neg \not\Rightarrow s_{cl}$) and a positive sentence can be in both ($s \Rightarrow s_{cl}, s \Rightarrow s_{cr}$).

$$s_\neg \not\Rightarrow s_{cl} \quad s_\neg \Rightarrow s_{cr} \qquad s \Rightarrow s_{cl} \quad s \Rightarrow s_{cr} \quad \text{and} \quad s_{cl} \not\Rightarrow s_{cr}$$

These constraints are lexically anchored by means of the lexical assignments below.

**Example 15.4** [Lexicon for *DL-Lite*$_{core}$ ] The lexicon entries to use are as below[28]

- Every $\in (s_t/(np\backslash s_{cr}))/n$: $\lambda X.\lambda Y.\forall x.X(x) \rightarrow Y(x)$

[27] We actually use residuated unary operators to carry out these derivability relations [KM95] exploiting their logical properties: $\Diamond_j \Box_j s \Rightarrow s \Rightarrow \Box_i \Diamond_i s$ etc. Examples of residuated unary operators are "possibility in the past" and "necessity in the future".

[28] Notice, in the present work we do not handle features of any sort (morphological etc). Their usage will make the lexical entries more complex but won't have any effect on the main idea we are presenting.

- is a $\in (np\backslash s)/n$: $\lambda X.\lambda z.X(z)$
- is not a $\in (np\backslash s_{\neg})/n$: $\lambda X.\lambda z.\neg X(z)$
- does not $\in (np\backslash s_{\neg})/(np\backslash s)$: $\lambda X.\lambda z.\neg X(z)$
- left $\in np\backslash s$: $\lambda z.\texttt{left}(z)$
- studies $\in np\backslash s$: $\lambda z.\exists x.\texttt{study}(z,x)$
- student $\in n$: $\lambda z.\texttt{student}(z)$
- everyone: $(s_t/(np\backslash s_{cr}))/(np\backslash s_{who})$: $\lambda X.\lambda Y.\forall x.X(x) \rightarrow Y(x)$
- who: $(np\backslash s_{who})/(np\backslash s_{cl})$, $\lambda P.\lambda z.P(z)$
- something: $((np\backslash s_{\exists})/np)\backslash(np\backslash s)$, $\lambda Z.\lambda y.\exists x.Z(y,x)$
- studies: $(np\backslash s_{\exists})/np$, $\lambda x.\lambda z.\texttt{studies}(z,x)$

First of all, notice that the categories assigned to *every* and *everyone* rule out the possibility for them to occur in object position –they can only be in a subject position. Moreover, since they are the only entries yielding a TBox sentence ($s_t$), only sentences starting with them will be considered as grammatical. The negation brings sentences to the negative sentential level, and once they are there they are blocked from occurring in the restrictive scope of *every* and *everyone*.

Finally, recall, that since, in this fragment we do not have the $\sqcap$ on the $Cl$ part, the introduction of the unqualified existential $\exists R$ in it can be performed only by means of the quantifier *everyone* followed by the relative pronoun "who" and a transitive verb composed with *something*. The introduction of $\exists R$ on the $Cr$ part correspond to the use of a transitive verb followed by an existential quantifier, *something*. The lexical entries for *everyone*, *who*, *something* and *studies* above account for these facts. The need of the $s_{who}$ categories is due to the fact that *everyone* must be followed by a relative clause, i.e., sentences like *everyone left* or *everyone walks and speaks* cannot be part of the grammar. Similarly, transitive verbs can occur on the $Cr$ part but only if followed by *something*, hence we use the category $s_{\exists}$ to guarantee this request.[29] Finally, the category assigned to "something" is such that it can occur only in object position.

The described fragment recognises as grammatical all the structures in (1)-(8) above.

The reader can gain a better understanding of the mechanisms involved by checking how the lexicon predicates the ungrammaticality of the sentences below whose meaning representations are not in *DL-Lite*.

(15) Everyone who does not know something left $[\neg\exists Know \sqsubseteq left]$

(16) Everyone who is not a boy left. $[\neg Boy \sqsubseteq left]$

We now move to *DL-Lite*$_{\mathcal{R},\sqcap}$, and account for the following additions

1. the conjunction can occur in the $Cl$ part (i.e., $Cl \longrightarrow Cl_1 \sqcap Cl_2$)
2. the qualified existential can occur in the $Cr$ part (i.e., $Cr \longrightarrow \exists r.A$

[29] Since we have neither $np$ nor $np/n$ entries we could also avoid the use of this extra sentential level $s_{\exists}$ in the sample example we are considering.

**Example 15.5** [Lexicon extension for *DL-Lite*$_{\mathcal{R},\sqcap}$ ] In order to move to *DL-Lite*$_{\mathcal{R},\sqcap}$, we need to add into the lexicon the following lexical entries.

- nice: $n_{cl}/n_{cl}$, $\lambda X.\lambda z.X(z) \wedge \texttt{nice}(z)$
- who: $(n_{cl}\backslash n_{cl})/(np\backslash s_{cl})$, $\lambda X.\lambda Y.\lambda z.X(x) \wedge Y(z)$
- and: $((np\backslash s_{cl})\backslash(np\backslash s_{cl}))/(np\backslash s_{cl})$, $\lambda X.\lambda Y.\lambda z.X(z) \wedge Y(z)$
- a: $(((np\backslash s_{\exists})/np)\backslash(np\backslash s_{cr}))/n$, $\lambda Y.\lambda Z.\lambda y.\exists x.Z(y,x) \wedge Y(x)$

Again, we use sentential levels to control the occurrence of these constructs. The extended lexicon accounts also for the structures in (9)-(14). Notice, the need of having the conjunction operating at the sentential level $s_{cl}$: this blocks the composition of negation (*does not*) with a verb phrase built by *and*, that would wrongly give: *does not walk and speak* with *not* having wide scope over *and*, viz. $\lambda z.\neg(\texttt{walk}(z) \wedge \texttt{speak}(z))$ that is not part of *DL-Lite*. For similar reasons, we have to block the composition of *is not a* with noun phrases built by means of an adjective. Again this composition would result into terms with the negation having scope over the conjunction, e.g., *is not a nice student* with term: $\lambda z.\neg(\texttt{nice}(z) \wedge \texttt{student}(z))$. The introduction of the category $n_{cl}$ with $n \Rightarrow n_{cl}$ helps blocking the construction of these terms. Furthermore, we have considered the version of *DL-Lite* with qualified existential of the form $\exists r.A$, rather than $\exists r.C$, hence the argument taken by the determiner *a* can only be a bare noun $n$.

Finally, notice, that the lexical entries for the adjective, conjunction and qualified existential are the ones that bring recursion into the language.

The fragment of sentences whose meaning representation belongs to a *DL-Lite* ABox is rather easy to build since an ABox consists only of unary or binary predicates whose arguments are constants. In other words, the lexicon is built only with noun, intransitive verbs, the copula (i.e., unary predicates), transitive verbs (i.e., binary predicates), and personal nouns. Since we can see any subset of ABox assertion as conjunction of such clauses, we could have in our lexicon also adjectives and relative pronoun.

## 15.5 Related Work

Our work is quite close to the research presented in [ST06] in that our Controlled Natural Language expressions have meaning representations that can be expressed in a DL. The difference lies on the one hand in the kind of DL we have considered, and on the other hand in the Grammar. With respect to the kind of DL, we have focused our attention on *DL-Lite*, which is a DL studied in the context of ontology-based access to (relational) databases [CDGL$^+$05a, CDGL$^+$05b]. As opposed to OWL-DL, which is the DL considered in [ST06], *DL-Lite* is specifically optimised with respect to the size of the data (rather than the size of the intensional descriptions in the ontology), when considering the trade-off between expressive power and computational complexity of inference. Indeed, it is, in a precise technical sense [CDGL$^+$06], the maximal DL that has the ability to efficiently and effectively manage very large data repositories by relying on industrial-strength relational database management systems (RDBMS). Moreover, *DL-Lite* can capture the essential features of the most commonly used formalisms for conceptual modeling, such as UML class diagrams and entity-relationship schemas [CDGL$^+$05a]. Hence, our work

is particularly relevant in all those contexts where NLIs to data-intensive systems need to be provided, and where the expressive power granted by richer DLs (e.g., OWL-DL) does not provide a sufficient guarantee for effectiveness. As for the formal grammar, we have used a logical grammar whose categories are recursively defined and are mapped to typed lambda terms. We believe this logical approach to parsing could help addressing the issue of defining also the fragments of natural language suitable for other relevant tasks of interest in the context of ontologies, such as querying or updating an ontology, or establishing mappings between different ontologies.

# References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[AL04] J. Angele and G. Lausen. Ontologies in F-logic. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 29–50. Springer Verlag, Berlin, Germany, 2004.

[Alh03] R. Alhajj. Extracting an extended entity-relationship model from a legacy relational database. *Information Systems*, 26(6):597–618, 2003.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[ALRR04] M. H. A. L. Rector, N. Drummond and J. E. Rogers. Owl pizzas: Common errors & common patterns from practical experience of teaching owl-dl. In *Proc. European Knowledge Acquisition Workshop (EKAW-2004).*, Aug. 2004.

[APTP03] K. Amir, S. Park, R. Tewari, and S. Padmanabhan. Scalable template-based query containment checking for web semantic caches. In *Proc. of the 19th IEEE Int. Conf. on Data Engineering (ICDE 2003)*, pages 493–504, 2003.

[ART95] I. Androutsopoulos, G. Ritchie, and P. Thanish. Natural language interface to databases – An introduction. *Natural Language Engineering*, 1:29–81, 1995. Cambridge University Press.

[ASU79a] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Trans. on Database Systems*, 4:297–314, 1979.

[ASU79b] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalence among relational expressions. *SIAM J. on Computing*, 8:218–246, 1979.

[Baa91] F. Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. In *Proc. of IJCAI-12*, Sydney, 1991.

[Baa96] F. Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.

[Baa03a] F. Baader. Computing the least common subsumer in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proceedings of the 11th International Conference on Conceptual Structures, ICCS 2003*, volume 2746 of *Lecture Notes in Artificial Intelligence*, pages 117–130. Springer-Verlag, 2003.

[Baa03b] 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, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 319–324. Morgan Kaufmann, 2003.

[Baa03c] F. Baader. Terminological cycles in a description logic with existential restrictions. In G. Gottlob and T. Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 325–330. Morgan Kaufmann, 2003.

[BBB+05] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet. Semantic Web Services Language (SWSL), September 2005. W3C member submission.

[BBL05] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *IJCAI-05*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[BBL07] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. *Artificial Intelligence*, 2007. Submitted.

[BBN+93] F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.

[BC02] P. Balbiani and J.-F. Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *Frontiers of Combining Systems (FroCoS 2002)*, number 2309 in LNAI, pages 162–176. Springer, 2002.

[BCM+03a] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge Univ. Press, Cambridge, UK, 2003.

[BCM+03b] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BCN92] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design. An Entity-Relationship Approach*. Benjamin/Cummings Publishing Company, Inc., 1992.

[Ben87] J. v. Benthem. Categorial grammar and lambda calculus. In D. Skordev, editor, *Mathematical Logic and its Applications*, pages 39–60. Plenum, New York, 1987.

[Ben97] B. Bennett. Modal logics for qualitative spatial reasoning. *J. of the Interest Group in Pure and Applied Logic*, 4(1), 1997.

[Ber02] R. Bernardi. *Reasoning with Polarity in Categorial Type Logic*. PhD thesis, UiL, OTS, Utrecht University, 2002.

[BFH+99] A. Borgida, E. Franconi, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Explaining alc subsumption. In *Proceedings of DL-99*, 1999.

[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.

[BGSS07] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*. AAAI Press, 2007.

[BH91] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[BH00] P. Burmeister and R. Holzer. On the treatment of incomplete knowledge in formal concept analysis. In *Proceedings of the 8th International Conference on Conceptual Structures, (ICCS 2000)*, volume 1867 of *Lecture Notes in Computer Science*, pages 385–398. Springer-Verlag, 2000.

[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*, pages 114–126. Springer-Verlag, 2005.

[BHGS01] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A Reasonable ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, number 2174 in Lecture Notes in Artificial Intelligence, pages 396–408. Springer, 2001. Appeared also in Proc. of the 2001 Description Logic Workshop (DL 2001).

[BHS02] F. Baader, I. Horrocks, and U. Sattler. Description logics for the semantic web. *KI – Künstliche Intelligenz*, 2002(4), 2002.

[BHS03a] F. Baader, I. Horrocks, and U. Sattler. Description logics. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks in Information Systems, pages 3–28. Springer–Verlag, Berlin, Germany, 2003.

[BHS03b] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In D. Hutter and W. Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.

[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 *Proc. of the 22nd German Annual Conf. on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1998.

[BKGK05] A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer. Querying ontologies: A controlled english interface for end-users. In *Proc. of the 4th Int. Semantic Web Conf. (ISWC 2005)*, pages 112–126, 2005.

[BKKK06] A. Bernstein, E. Kaufmann, C. Kaiser, and C. Kiefer. Ginseng: A guided input natural language search engine for querying ontologies. In *2006 Jena User Conference*, May 2006.

[BKM99] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 96–101, 1999.

[BKT02a] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.

[BKT02b] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 203–214, San Francisco, CA, 2002. Morgan Kaufman.

[BL84] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pages 34–37, 1984.

[BLS06] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.

[Bon04] P. A. Bonatti. On the decidability of containment of recursive datalog queries - preliminary report. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004)*, pages 297–306, 2004.

[Bra04] S. Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In R. L. de Mantáras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, pages 298–302. IOS Press, 2004.

[BS95] P. Blackburn and J. Seligman. Hybrid languages. *J. of Logic, Language and Information*, 4:251–272, 1995.

[BS04] F. Baader and B. Sertkaya. Applying formal concept analysis to description logics. In P. Eklund, editor, *Proceedings of the 2nd International Conference on Formal Concept Analysis (ICFCA 2004)*, volume 2961 of *Lecture Notes in Computer Science*, pages 261–286, Sydney, Australia, 2004. Springer-Verlag.

[BST04a] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, CEUR-WS, 2004.

[BST04b] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In J. Alferes and J. Leite, editors, *Proc. of the 9th Eur. Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 400–412, Lisbon, Portugal, 2004. Springer-Verlag.

[BST07] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Applied Logic*, 2007. To be published.

[BTK03] S. Brandt, A.-Y. Turhan, and R. Küsters. Extensions of non-standard inferences for description logics with transitive roles. In M. Vardi and A. Voronkov, editors, *Proc. of the 10th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2003)*, volume 2850 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[Cal06] A. Calì. Containment of conjunctive queries over conceptual schemata. In *The 11th International Conference on Database Systems for Advanced Applications (DASFAA 2006)*, pages 628–643, April 2006.

[CBS94] R. H. L. Chiang, T. M. Barron, and V. C. Storey. Reverse engineering of relational databases: extraction of an eer model from a relational database. *Data and Knowledge Engineering*, 12(2):107–142, 1994.

[CDGL$^+$] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Submitted.

[CDGL97] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment in Description Logics with $n$-ary relations. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 5–9, 1997.

[CDGL98a] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[CDGL+98b] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[CDGL00] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[CDGL+01] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int. J. of Cooperative Information Systems*, 10(3):237–271, 2001.

[CDGL02] D. Calvanese, G. De Giacomo, and M. Lenzerini. Description logics for information integration. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*, volume 2408 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2002.

[CDGL+05a] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[CDGL+05b] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tailoring OWL for data intensive ontologies. In *Proc. of the Workshop on OWL: Experiences and Directions (OWLED 2005)*, 2005.

[CDGL+06] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[CDGL07] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. on Computational Logic*, 2007. To appear.

[CDGV03] D. Calvanese, G. De Giacomo, and M. Y. Vardi. Decidable containment of recursive queries. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, volume 2572 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2003.

[CGG+06] D. Calvanese, B. C. Grau, G. D. Giacomo, E. Franconi, I. Horrocks, A. Kaplunova, D. Lembo, C. Lutz, D. Martinenghi, R. Möller, R. Rosati, S. Tessaris, and A.-Y. Turhan. Common framework for representing ontologies. Deliverable D08, TONES EU-IST STREP FP6-7603, August 2006.

[CGL+06] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th*

*Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[CH94] W. W. Cohen and H. Hirsh. Learning the CLASSIC description logics: Theorethical and experimental results. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 121–133, 1994.

[Cha92] E. P. F. Chan. Containment and minimization of positive conjunctive queries in OODB's. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 202–211, 1992.

[CK06] A. Calì and M. Kifer. Containment of conunctive object meta-queries. In *International Conference on Very Large Data Bases*, pages 942–952, 2006.

[CL93] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[CLN98] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publishers, 1998.

[CM77] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77)*, pages 77–90, 1977.

[CR97] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 56–70, 1997.

[CR03] D. Calvanese and R. Rosati. Answering recursive queries under keys and foreign keys is undecidable. In *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-79/`, 2003.

[CV92] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 55–66, 1992.

[CV99] R. Casati and A. Varzi. *Parts and Places. The Structure of Spatial Representation*. MIT Press, Cambridge, MA, 1999.

[dB05] J. de Bruijn. The WSML specification, February 2005.

[DF06] P. Dongilli and E. Franconi. An intelligent query interface with natural language support. In *Proc. of the 19th Int. Florida Artificial Intelligence Research Society Conference (FLAIRS 2006)*, May 2006.

[DFT04] P. Dongilli, E. Franconi, and S. Tessaris. Semantics driven support for query formulation. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, volume 104, June 2004.

[DG84] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. of Logic Programming*, 1(3):267–284, 1984.

[DK06] D. Dinh-Khac. Two types of key constraints in description logics with concrete domains. Master thesis, TU Dresden, Germany, 2006. See http://lat.inf.tu-dresden.de/research/papers.html.

[DPF+05] L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng, and P. Kolari. Finding and Ranking Knowledge on the Semantic Web. In *Proc. of the 4th International Semantic Web Conference*, LNCS 3729, pages 156–170. Springer, November 2005.

[DS96] G. Dong and J. Su. Conjunctive query containment with respect to views and constraints. *Information Processing Lett.*, 57(2):95–102, 1996.

[EF91] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[Eij85] J. v. Eijck. *Aspects of Quantification in Natural Language*. PhD thesis, University of Groningen, 1985.

[EN04] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley Publ. Co., fourth edition, 2004.

[FHL+98] J. Frohn, R. Himmerder, G. Lausen, W. May, and C. Schlepphorst. Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23(8):589–613, 1998.

[FLOa] FLORA-2. The FLORA-2 Web site. http:// flora.sourceforge.net.

[FLOb] FLORID. The FLORID system. http:// www.informatik.uni-freiburg.de/∼dbis/florid/.

[FP96] M. Frazier and L. Pitt. CLASSIC learning. *Machine Learning*, 25:151–193, 1996.

[Fra96] A. Frank. Qualitative spatial reasoning: Cardinal directions as an example. *International Journal of Geographical Information Systems*, 10(3):269–290, 1996.

[FTG+06] E. Franconi, S. Tessaris, B. C. Grau, B. Suntisrivaraporn, C. Lutz, R. Moller, and D. Lembo. Draft ontology task handbook. Deliverable D03, TONES EU-IST STREP FP6-7603, March 2006.

[Gan84] B. Ganter. Two basic algorithms in concept analysis. Technical Report Preprint-Nr. 831, Technische Hochschule Darmstadt, Darmstadt, Germany, 1984.

[Gen00] T. Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[GHKS07] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. A logical framework for modularity of ontologies. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*. AAAI Press, 2007.

[GHLS07] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 2007.

[GL94] G. D. Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94). Volume 1*, pages 205–212. AAAI Press, 1994.

[GLL07] S. Goeller, M. Lohrey, and C. Lutz. Pdl with intersection and converse is 2exp-complete. In H. Seidl, editor, *Tenth International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2007)*, Lecture Notes in Computer Science. Springer, 2007.

[GLW06] S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? a case for conservative extensions in description logics. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 187–197. AAAI Press, 2006.

[GM95] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 18(2):3–18, 1995.

[GMF+03] J. Gennari, M. Musen, R. Fergerson, W. Grosse, M. Crubezy, H. Eriksson, N. Noy, and S. Tu. The evolution of protege: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[Gro06] O. M. Group. *Meta Object Facility (MOF) Core Specification*. Object Management Group, `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`, 2006.

[GSUW94] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proc. of the 13th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'94)*, 1994.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, Germany, 1999.

[Hai98] J. L. Hainaut. Database reverse engineering: models, techniques and strategies. In *Proc. of the 10th Conference on ER Approach*, 1998.

[Hal01] A. Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.

[HH01] J. Heflin and J. Hendler. A portrait of the semantic web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[HLM98] V. Haarslev, C. Lutz, and R. Möller. Foundations of spatioterminological reasoning with description logics. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 112–123, 1998.

[HM01] V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[HMW01] V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In *Proceedings of the First International Joint Conference on Automated Reasoning IJCAR'01*, pages 29–44, 2001.

[Hof05] M. Hofmann. Proof-theoretic approach to description-logic. In *Proceedings of Logic in Computer Science (LICS05)*, pages 229–237. IEEE Computer Society, 2005.

[Hol04a] R. Holzer. Knowledge acquisition under incomplete knowledge using methods from formal concept analysis: Part I. *Fundamenta Informaticae*, 63(1):17–39, 2004.

[Hol04b] R. Holzer. Knowledge acquisition under incomplete knowledge using methods from formal concept analysis: Part II. *Fundamenta Informaticae*, 63(1):41–63, 2004.

[Hor98] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[HPS99] I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[HPSvH03] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[HS99] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[HS01] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.

[HS03a] J. Hladik and U. Sattler. A translation of looping alternating automata to description logics. In *Proc. of the 19th Conference on Automated Deduction (CADE-19)*, volume 2741 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2003.

[HS03b] I. Horrocks and U. Sattler. Decidability of shiq with complex role inclusion axioms. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 343–348. Morgan-Kaufmann Publishers, 2003.

[HS04] I. Horrocks and U. Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, 160:79–104, 2004.

[HS05] I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, Edinburgh (UK), 2005. Morgan Kaufmann.

[HST99] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[HST00] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In D. MacAllester, editor, *Proc. of CADE-17*, volume 1831 of *LNCS*, Germany, 2000. Springer.

[HT00] I. Horrocks and S. Tessaris. A conjunctive query language for description logic ABoxes. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.

[HU97] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[Huj98] W. O. Hujsen. Controlled language – an introduction. In *Proc. of the 2nd Int. Workshop on Controlled Language Applications (CLAW 1998)*, pages 1–15, Pittsburg, 1998.

[Hul97] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 51–61, 1997.

[Hut06] M. Huth. Some current issues in model checking. *Software Tools for Technology Transfer*, 8(4):1–10, 2006.

[Jac00] D. Jackson. Automating first-order relational logic. In *SIGSOFT '00/FSE-8: Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 130–139, New York, NY, USA, 2000. ACM Press.

[Jac06] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[JK84] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[JS99] L. P. D. Jesus and P. Sousa. Selection of reverse engineering methods for relational databases. In *Proc. of the 3rd Int. Conf. on Software Maintenance and Reengineering*, pages 194–197, 1999.

[KB01] R. Küsters and A. Borgida. What's in an attribute? Consequences for the least common subsumer. *J. of Artificial Intelligence Research*, 14:167–203, 2001.

[KdN03] Y. Kazakov and H. de Nivelle. Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSpace-complete. In E. F. Diego Calvanese, Giuseppe De Giacomo, editor, *Proceedings of the International Workshop in Description Logics 2003 (DL2003)*, number 81 in CEUR-WS (http://ceur-ws.org/), 2003.

[KF85] E. Keenan and L. Faltz. *Boolean Semantics for Natural Language*. Reidel, Dordrecht, 1985.

[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 *Proceedings of the Third International Semantic Web Conference*, Hiroshima, Japan, 2004.

[Kif05] M. Kifer. Rules and ontologies in f-logic. In *Reasoning Web*, number 3564 in Lecture Notes in Computer Science, pages 22–34, July 2005.

[Kit03] R. I. Kittredge. Sublanguages and controlled languages. In R. Mitkov, editor, *The Oxford Handbook of Computational Linguistics*, pages 430–447. Oxford University Press, 2003.

[KL89] M. Kifer and G. Lausen. F-Logic: A higher-order language for reasoning about objects, inheritance and schema. In *ACM SIGMOD Conference on Management of Data*, pages 134–146, New York, 1989. ACM.

[Klu88] A. C. Klug. On conjunctive queries containing inequalities. *J. of the ACM*, 35(1):146–160, 1988.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[KM95] N. Kurtonina and M. Moortgat. Structural control. In P. Blackburn and M. de Rijke, editors, *Logic, Structures and Syntax*. Dordrecht: Kluwer, 1995.

[KM01] R. Küsters and R. Molitor. Computing least common subsumers in $\mathcal{ALEN}$. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 219–224, 2001.

[Kod04] V. Kodaganallur. Incorporating language processing into Java applications: A JavaCC tutorial. *IEEE Software*, 21(4):70–77, 2004.

[KPS+05] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca-Grau, and J. Hendler. Swoop - a web ontology editing browser. *Journal of Web Semantics*, 1(4), 2005.

[KPSCG06] A. Kalyanpur, B. Parsia, E. Sirin, and B. Cuenca-Grau. Repairing unsatisfiable concepts in owl ontologies. In *Proceedings of the European Semantic Web Conference (ESWC'06)*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184. Springer, 2006.

[KPSH05] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable classes in owl ontologies. *Journal of Web Semantics*, 2005. To Appear.

[Küs01] R. Küsters. *Non-standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.

[LAHS05] C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains. *J. of Artificial Intelligence Research*, 23:667–726, 2005.

[Lam58] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[Lam06] L. Lamport. The +CAL algorithm language. Technical report, Microsoft, 2006.

[LBT92] A. Lefebvre, P. Bernus, and R. Topor. Query transformation for accessing heterogeneous databases. In *Proceedings of the JICSLP-92 Workshop on Deductive Databases*, November 1992.

[Le99] O. Lasilla and R. S. (editors). Resource description framework (RDF) model and syntax specification. Technical report, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[Len02] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS02)*, pages 233–346, 2002.

[LH03] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Twelfth International World Wide Web Conference (WWW 2003)*, 2003.

[LL01] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[LLS06] D. Lembo, C. Lutz, and B. Suntisrivaraporn. Tasks for ontology design and maintenance. Deliverable D05, TONES EU-IST STREP FP6-7603, May 2006.

[LM04] C. Lutz and M. Milicic. Description logics with concrete domains and functional dependencies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, 2004.

[LM07] C. Lutz and M. Milicic. A tableau algorithm for description logics with concrete domains and general tboxes. *Journal of Automated Reasoning. Special Issue on Automated Reasoning with Analytic Tableaux and Related Methods*, 2007. To appear.

[LR96] A. Y. Levy and M.-C. Rousset. CARIN: A representation language combining Horn rules and description logics. In *Proc. of the 12th Eur. Conf. on Artificial Intelligence (ECAI'96)*, pages 323–327, 1996.

[LS97] A. Y. Levy and D. Suciu. Deciding containment for queries with complex objects. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 20–31, 1997.

[LSK95] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.

[Lut01] C. Lutz. NEXPTIME-complete description logics with concrete domains. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 45–60. Springer, 2001.

[Lut02a] C. Lutz. Adding numbers to the $\mathcal{SHIQ}$ description logic—First results. In *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 191–202. Morgan Kaufmann, 2002.

[Lut02b] C. Lutz. Reasoning about entity relationship diagrams with complex attribute dependencies. In I. Horrocks and S. Tessaris, editors, *Proceedings of the International Workshop in Description Logics 2002 (DL2002)*, number 53 in CEUR-WS (http://ceur-ws.org/), pages 185–194, 2002.

[Lut03] C. Lutz. Description logics with concrete domains—a survey. In P. Balbiani, N.-Y. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Advances in Modal Logics Volume 4*, pages 265–296. King's College Publications, 2003.

[Lut04a] C. Lutz. Combining interval-based temporal reasoning with general TBoxes. *Artificial Intelligence*, 152(2):235–274, 2004.

[Lut04b] C. Lutz. NExpTime-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 5(4):669–705, 2004.

[Lut06] C. Lutz. Complexity and succinctness of public announcement logic. In P. Stone and G. Weiss, editors, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, pages 137–144. Association for Computing Machinery (ACM), 2006.

[LWW07] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*. AAAI Press, 2007.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[Mai97] T. Maibaum. Conservative extensions, interpretations between theories and all that! In *Proc. of 7th CAAP/FASE Conf. on Theory and Practice of Software Development*, pages 40–66, 1997.

[McG96] D. L. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Department of Computer Science, Rutgers University, Oct. 1996. Also available as Rutgers Technical Report Number LCSR-TR-277.

[MLF00] T. Millstein, A. Levy, and M. Friedman. Query containment for data integration systems. In *ACM Symposium on Principles of Database Systems*, pages 67–75, New York, NY, USA, 2000. ACM Press.

[MM90] V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, 1990.

[MMS79] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.

[Moo97] M. Moortgat. Categorial Type Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 93–178. The MIT Press, Cambridge, Massachusetts, Cambridge, 1997.

[Moo98] R. Moot. Grail: An automated proof assistant for categorial grammar logics. In R. Backhouse, editor, *Proc. of the 1998 User Interfaces for Theorem Provers Conference*, pages 120–129, 1998.

[MS06] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Proc. of the 13th International Conference on Logic for Programming, Artificial Intelligence (LPAR06)*, LNCS. Springer Verlag, 2006.

[NB95] B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[Neb88] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[Neb90] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[NS03] F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, pages 315–329, 2003.

[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*, pages 314–325. Springer-Verlag, 2002.

[OCE06] M. M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of answering unions of conjunctive queries in $\mathcal{SHIQ}$. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[Ont] Ontoprise, GmbH. Ontobroker. http:// www.ontoprise.com/.

[OVSM04] D. Oberle, R. Volz, S. Staab, and B. Motik. An extensible ontology software environment. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 299–320. Springer-Verlag, 2004.

[PHT06] I. Pratt-Hartmann and A. Third. More fragments of language: the case of ditransitive verbs. *Notre Dame Journal of Formal Logic*, 47(2), 2006.

[PS05] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2005.

[PSK05] B. Parsia, E. Sirin, and A. Kalyanpur. Debugging OWL ontologies. In A. Ellis and T. Hagino, editors, *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*, pages 633–640. ACM, 2005.

[RCC92] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176. Morgan Kaufman, 1992.

[RH97] A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, Stanford, CA, 1997. AAAI Press.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[RN99] J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1–2):69–123, 1999.

[SC03a] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of IJCAI, 2003*, 2003.

[SC03b] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies (extended abstract). In *Proceedings of the 15th Belgium-Netherlands Conference on Artificial Intelligence*, 2003.

[Sch91] K. Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. of IJCAI-12*, pages 466–471, Sydney, 1991.

[Sch94] A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[Sch05a] S. Schlobach. Debugging and semantic clarification by pinpointing. In A. Gomez-Perez and J. Euzenat, editors, *Proceedings of the European Semantic Web Conference (ESWC05)*, pages 226–240. Springer, 2005.

[Sch05b] S. Schlobach. Diagnosing terminologies. In M. M. Veloso and S. Kambhampati, editors, *Proceedings of The Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, pages 670–675. AAAI Press/The MIT Press, 2005.

[SD02] M. Sintek and S. Decker. TRIPLE – A query, inference, and transformation language for the Semantic Web. In *International Semantic Web Conference (ISWC)*, June 2002.

[Sim87] P. Simons. *Parts: A Study in Ontology*. Clarendon Press, Oxford, 1987.

[Sow04] J. F. Sowa. Common logic controlled english. Draft, 24 February 2004.

[Spa00] K. A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the American Medical Informatics Association*, 2000. Fall Symposium Special Issue.

[SS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[ST04a] R. Schwitter and M. Tilbrook. Controlled natural language meets the semantic web. In S. W. A. Asudeh, C. Paris, editor, *Proc. of the Australasian Language Technology Workshop 2004*, pages 55–62. Macquarie University, 2004.

[ST04b] R. Schwitter and M. Tilbrook. Dynamic semantics at work. In *Proc. of the Int. Workshop on Logic and Engineering of Natural Language Semantics*, pages 49–60, May 2004.

[ST06] R. Schwitter and M. Tilbrook. Let's talk in description logic via controlled natural language. In *Proc. of the 3rd Int. Workshop on Logic and Engineering of Natural Language Semantics (LENLS 2006)*, pages 193–207, 2006.

[SY80] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. of the ACM*, 27(4):633–655, 1980.

[Tho74] R. Thomason, editor. *Formal Philosophy: Selected papers of Richard Montague*. Yale University Press, New Haven, 1974.

[TJ06] E. Torlak and D. Jackson. The Design of a Relational Engine. Technical Report MIT-CSAIL-TR-2006-068, MIT CSAIL, 2006.

[TK04] A.-Y. Turhan and C. Kissig. SONIC — Non-standard inferences go OILED. In D. Basin and M. Rusinowitch, editors, *Proc. of the 2nd Int. Joint Conf. on Automated Reasoning (IJCAR 2004)*, volume 3097 of *Lecture Notes in Computer Science*, pages 321–325. Springer-Verlag, 2004. SONIC is available from `http://wwwtcs.inf.tu-dresden.de/~sonic/`.

[TLN06] T. Teorey, S. Lightstone, and T. Nadeau. *Database Modeling and Design*. Morgan Kaufmann Publishers, fourth edition, 2006.

[Tob01] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[Ull97] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.

[vdM98] R. van der Meyden. Logical approaches to incomplete information. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publishers, 1998.

[vEB97] P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, Recursion Theory*. 1997.

[VK86] M. B. Vilain and H. A. Kautz. Constraint propagation algorithms for temporal reasoning. In *AAAI*, pages 377–382, 1986.

[VKvB90] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: a revised report. In *Readings in qualitative reasoning about physical systems*, pages 373–381. Morgan Kaufmann, San Francisco, CA, USA, 1990.

[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*, pages 786–800. Springer-Verlag, 2005.

[WDSS06] H. Wang, J. S. Dong, J. Sun, and J. Sun. Reasoning support for semantic web ontology family languages using alloy. *International Journal of Multiagent and Grid Systems, Special issue on Agent-Oriented Software Development Methodologies*, 2(4), December 2006.

[Wid95] J. Widom (ed.). Special issue on materialized views and data warehousing. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 18(2), 1995.

[Woo03] P. T. Wood. Containment for XPath fragments under DTD constraints. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, pages 300–314, 2003.

[WVV+01] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-based integration of information - a survey of existing approaches. In *Proc. of IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.

[YKZ03] G. Yang, M. Kifer, and C. Zhao. FLORA-2: A rule-based knowledge representation and inference infrastructure for the Semantic Web. In *International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-2003)*, November 2003.