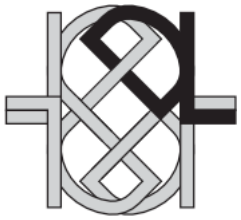
# Proceedings of the

## 2004 International Workshop on Description Logics (DL2004)



Whistler, BC, Canada
June 6-8, 2004

Edited by:

Volker Haarslev
Ralf Möller

Also electronically available as CEUR Publication at
http://CEUR-WS.org

# Preface

The 2004 International Workshop on Description Logics (DL'04) was held in Whistler, British Columbia, Canada, from June 6 to 8, 2004. It continued the tradition of international workshops devoted to discussing developments and applications of knowledge representation formalisms based on Description Logics. The list of International Workshops on Description Logics can be found at http://dl.kr.org.

Each paper submitted to DL'04 was reviewed by at least two members of the program committee, or by additional reviewers recruited by the PC members. Due to the success of the poster session from the previous DL workshop, it was decided to schedule a poster session together with a set of system demonstrations that were often motivated by the advent of the semantic web.

In addition to the presentation of accepted papers, posters, and demonstrations, two invited talks were presented at DL'04:

- Sheila McIlraith, University of Toronto, Toronto, Canada, gave a talk on OWL-S and web agent/web service composition;
- Frank Wolter, University of Liverpool, Liverpool, UK, gave a talk on Combining Description Logics.

We would like to thank the Concordia University and the Technical University Hamburg-Harburg for their support.

Volker Haarslev
Ralf Möller
(The DL'04 PC chairs)

# DL2004 Program Committee

Carlos Areces
INRIA Lorraine, France

Diego Calvanese
Faculty of Computer Science, Free University of Bozen-Bolzano

Enrico Franconi
Faculty of Computer Science, Free University of Bozen-Bolzano

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"

Volker Haarslev
Computer Science Department, Concordia University, Montreal, Quebec, Canada

Ian Horrocks
Department of Computer Science, University of Manchester, Manchester, UK

Ralf Küsters
Department of Computer Science, Stanford University, CA, USA

Carsten Lutz
Technical University Dresden, Department of Computer Science, Institute for Theoretical Computer Science, Dresden, Germany

Deborah L. McGuinness
Knowledge Systems Laboratory, Stanford University, CA, USA

Ralf Möller
Technical University Hamburg-Harburg, Germany

Peter Patel-Schneider
Network Data and Services Research Department, Bell Laboratories, Murray Hill, NJ, USA

Ulrike Sattler
Department of Computer Science, University of Manchester, Manchester, UK

Grant Weddell
Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada

Chris Welty
IBM Watson Research Center, Hawthorne, NY, USA

## Additional reviewers

Franz Baader, Sebastian Brandt, Dimitry Tsarkov, Anni-Yasmin Turhan, Daniele Turi.

# Contents

## Theory 1



## Implementation Techniques 1



## Graphical Interfaces 1

## Implementation Techniques 2



## Applications 1



## Posters



## Applications 2

## Theory 2



## Graphical Interfaces 2



## Position Papers

# Reasoning on Temporal Conceptual Schemas with Dynamic Constraints

Alessandro Artale*
Faculty of Computer Science – Free University of Bozen-Bolzano
artale@inf.unibz.it

## 1 Introduction

Temporally enhanced conceptual models have been developed to help designing temporal databases [12]. In this paper we deal with Extended Entity-Relationship (EER) diagrams[1] used to model temporal databases. The temporal conceptual model $\mathcal{ER}_{VT}$ has been introduced both to *formally* clarify the meaning of the various temporal constructs appeared in the literature [2, 4], and to check the possibility to perform *reasoning* on top of temporal schemas [5]. $\mathcal{ER}_{VT}$ supports valid time for entities, attributes, and relationships in the line of TIMEER [10] and ERT [15], while supporting dynamic constraints for entities as presented in MADS [14]. $\mathcal{ER}_{VT}$ is able to distinguish between *snapshot* constructs—i.e. each of their instances has a global lifespan—and *temporary* constructs—i.e. each of their instances have a limited lifespan. Dynamic constructs capture the *object migration* from a source entity to a target entity.

The contribution of this paper is twofold. Moving from the formal characterization of $\mathcal{ER}_{VT}$ given in [4] we clarify the relevant reasoning problems for temporal EER diagrams. In particular, we distinguish between six different reasoning services, introducing two new services for both entities and relationships: *liveness satisfiability*—i.e. whether an entity or relationship admits a non-empty extension infinitely often in the future—and *global satisfiability*—i.e. whether an entity or relationship admits a non-empty extension at all points in time. After a systematic definition of the various reasoning problems we then show that all the satisfiability problems (i.e. schema, entity and relationship satisfiability problems) together with the subsumption problem (i.e. checking whether two entities or relationships denote one a subset of the other so that there is an implicit ISA link between them) can be mutually reduced to each other. On the other hand, checking whether a schema *logically implies* another schema is shown to be the more general reasoning service.

The second contribution is to prove that reasoning on temporal conceptual models is undecidable provided the diagrams are able to: (a) Distinguish between temporal and non-temporal constructs; (b) Represent *dynamic constraints* between entities, i.e. entities whose instances

*The author has been partially supported by the EU projects Sewasie, KnowledgeWeb, and Interop. This paper is a shorter version of [1].

[1] EER is the standard entity-relationship data model, enriched with ISA links, generalized hierarchies with disjoint and covering constraints, and full cardinality constraints.

| | | | | | |
|---|---|---|---|---|---|
| $C, D$ | $\rightarrow$ | $A \mid$ | (atomic concept) | $A^{\mathcal{I}(t)} \subseteq$ | $\Delta^{\mathcal{I}}$ |
| | | $\top \mid$ | (top) | $\top^{\mathcal{I}(t)} =$ | $\Delta^{\mathcal{I}}$ |
| | | $\bot \mid$ | (bottom) | $\bot^{\mathcal{I}(t)} =$ | $\emptyset$ |
| | | $\neg C \mid$ | (complement) | $(\neg C)^{\mathcal{I}(t)} =$ | $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}(t)}$ |
| | | $C \sqcap D \mid$ | (conjunction) | $(C \sqcap D)^{\mathcal{I}(t)} =$ | $C^{\mathcal{I}(t)} \cap D^{\mathcal{I}(t)}$ |
| | | $C \sqcup D \mid$ | (disjunction) | $(C \sqcup D)^{\mathcal{I}(t)} =$ | $C^{\mathcal{I}(t)} \cup D^{\mathcal{I}(t)}$ |
| | | $\exists R.C \mid$ | (exist. quantifier) | $(\forall R.C)^{\mathcal{I}(t)} =$ | $\{a \in \Delta^{\mathcal{I}} \mid \forall b.R^{\mathcal{I}(t)}(a,b) \Rightarrow C^{\mathcal{I}(t)}(b)\}$ |
| | | $\forall R.C \mid$ | (univ. quantifier) | $(\exists R.C)^{\mathcal{I}(t)} =$ | $\{a \in \Delta^{\mathcal{I}} \mid \exists b.R^{\mathcal{I}(t)}(a,b) \wedge C^{\mathcal{I}(t)}(b)\}$ |
| | | $\Diamond^+ C \mid$ | (Sometime) | $(\Diamond^+ C)^{\mathcal{I}(t)} =$ | $\{a \in \Delta^{\mathcal{I}} \mid \exists v > t.C^{\mathcal{I}(v)}(a)\}$ |
| | | $\Box^+ C \mid$ | (Every time) | $(\Box^+ C)^{\mathcal{I}(t)} =$ | $\{a \in \Delta^{\mathcal{I}} \mid \forall v > t.C^{\mathcal{I}(v)}(a)\}$ |

Figure 1: Syntax and Semantics for the $\mathcal{ALC}_{\mathsf{F}}$ Description Logic

migrate to other entities. To the best of our knowledge, this is the first time such a result is proved. Indeed, the result presented in [5] showed that $\mathcal{ER}_{VT}$ diagrams can be embedded into the temporal description logic (DL) $\mathcal{DLR}_{\mathcal{US}}$—where $\mathcal{U}$, $\mathcal{S}$ extend $\mathcal{DLR}$ with the *until* and *since* temporal modalities—and that reasoning in $\mathcal{DLR}_{\mathcal{US}}$ was undecidable. Instead, here we prove that even reasoning just on $\mathcal{ER}_{VT}$ schemas is undecidable. The undecidability result is proved via a reduction of the Halting Problem with a technique similar to [9]. In particular, we proceed by first showing that the halting problem can be encoded as a Knowledge Base (KB) in $\mathcal{ALC}_{\mathsf{F}}$—where $\mathsf{F}$ extends $\mathcal{ALC}$ with the *future* temporal modality—and then proving that such a KB in $\mathcal{ALC}_{\mathsf{F}}$ can be captured by an $\mathcal{ER}_{VT}$ diagram.

The paper is organized as follows. The temporal DL $\mathcal{ALC}_{\mathsf{F}}$ and the conceptual model $\mathcal{ER}_{VT}$ are formally presented in Sections 2 and 3, respectively. The various reasoning services for temporal conceptual modeling are defined in Section 4 and their equivalence is proved. That reasoning in presence of dynamic constraints is undecidable is proved in Section 5.

## 2 The Temporal Description Logic

In this Section we introduce the $\mathcal{ALC}_{\mathsf{F}}$ DL [16, 3, 9] as a the tense-logical extension of $\mathcal{ALC}$. Basic types of $\mathcal{ALC}_{\mathsf{F}}$ are *concepts* and *roles*. According to the syntax rules of Figure 1, $\mathcal{ALC}_{\mathsf{F}}$ *concepts* are built out of *atomic concepts* and *atomic roles*. Tense operators are added for concepts: $\Diamond^+$ (sometime in the future) and $\Box^+$ (always in the future). Furthermore, while tense operators are allowed only at the level of concepts—i.e. no temporal operators are allowed on roles—we will distinguish between so called *local*—$\mathcal{RL}$—and *global*—$\mathcal{RG}$—roles.

Let us now consider the formal semantics of $\mathcal{ALC}_{\mathsf{F}}$. A temporal structure $\mathcal{T} = (\mathcal{T}_p, <)$ is assumed, where $\mathcal{T}_p$ is a set of time points and $<$ is a strict linear order on $\mathcal{T}_p$—$\mathcal{T}$ is assumed to be isomorphic to either $(\mathbb{Z}, <)$ or $(\mathbb{N}, <)$. An $\mathcal{ALC}_{\mathsf{F}}$ *temporal interpretation* over $\mathcal{T}$ is a triple of the form $\mathcal{I} \doteq \langle \mathcal{T}, \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}(t)} \rangle$, where $\Delta^{\mathcal{I}}$ is non-empty set of objects and $\cdot^{\mathcal{I}(t)}$ an *interpretation function* such that, for every $t \in \mathcal{T}$, every concept $C$, and every role $R$, we have $C^{\mathcal{I}(t)} \subseteq \Delta^{\mathcal{I}}$ and $R^{\mathcal{I}(t)} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Furthermore, if $R \in \mathcal{RG}$, then, $\forall t_1, t_2 \in \mathcal{T}.R^{\mathcal{I}(t_1)} = R^{\mathcal{I}(t_2)}$. The semantics of $\mathcal{ALC}_{\mathsf{F}}$ concepts is defined in Figure 1.

A *knowledge base* (KB) in this context is a finite set $\Sigma$ of *terminological axioms* of the form $C \sqsubseteq D$. An interpretation $\mathcal{I}$ satisfies $C \sqsubseteq D$ iff the interpretation of $C$ is included in the interpretation of $D$ at all time, i.e. $C^{\mathcal{I}(t)} \subseteq D^{\mathcal{I}(t)}$, for all $t \in \mathcal{T}$. A knowledge base $\Sigma$ is *satisfiable* if there is a temporal interpretation $\mathcal{I}$ that satisfies every axiom in $\Sigma$. $\Sigma$ *logically implies* an axiom $C \sqsubseteq D$ (written $\Sigma \models C \sqsubseteq D$) if $C \sqsubseteq D$ is satisfied by every model of $\Sigma$. A concept $C$ is satisfiable, given a knowledge base $\Sigma$, if there exists a model $\mathcal{I}$ of $\Sigma$ such that $C^{\mathcal{I}(t)} \neq \emptyset$ for some $t \in \mathcal{T}$, i.e. $\Sigma \not\models C \sqsubseteq \bot$.

# 3 Temporal Conceptual Modeling

In this Section, the temporal EER model $\mathcal{ER}_{VT}$ is briefly introduced. $\mathcal{ER}_{VT}$ supports valid time for entities, attributes, and relationships in the line of TIMEER [10] and ERT [15], while supporting dynamic constraints for entities as presented in MADS [14]. $\mathcal{ER}_{VT}$ is able to distinguish between *snapshot* (see the consensus glossary [11] for the terminology used) constructs—i.e. each of their instances has a global lifespan—*temporary* constructs—i.e. each of their instances have a limited lifespan—or *implicitly temporal* constructs—i.e. their instances can have either a global or a temporary existence. Two temporal marks, S (snapshot) and VT (valid time), are introduced in $\mathcal{ER}_{VT}$ to capture such temporal behavior.

Dynamic constructs capture the *object migration* from a source entity to a target entity. If there is a *dynamic extension* between a source and a target entity (represented in $\mathcal{ER}_{VT}$ by a dotted link labeled with DEX) models the case where instances of the source entity *eventually* become instances of the target entity. On the other hand, a *dynamic persistency* (represented in $\mathcal{ER}_{VT}$ by a dotted link labeled with PER) models the dual case of instances *persistently* migrating to a target entity (for a complete introduction on $\mathcal{ER}_{VT}$ with a worked out example see [4]).

$\mathcal{ER}_{VT}$ is equipped with both a linear and a graphical syntax along with a model-theoretic semantics as a temporal extension of the EER semantics [7]. Presenting the $\mathcal{ER}_{VT}$ linear syntax, we adopt the following notation: given two sets $X, Y$, an *X-labeled* tuple over $Y$ is a function from $X$ to $Y$; the labeled tuple $T$ that maps the set $\{x_1, \ldots, x_n\} \subseteq X$ to the set $\{y_1, \ldots, y_n\} \subseteq Y$ is denoted by $\langle x_1 : y_1, \ldots, x_n : y_n \rangle$, and $T[x_i] = y_i$. An $\mathcal{ER}_{VT}$ schema is a tuple:

$$\Sigma = (\mathcal{L}, \text{REL}, \text{ATT}, \text{CARD}, \text{ISA}, \text{DISJ}, \text{COVER}, \text{S}, \text{T}, \text{KEY}, \text{DEX}, \text{PER}), \text{ such that:}$$

$\mathcal{L}$ is a finite alphabet partitioned into the sets: $\mathcal{E}$ (*entity* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{R}$ (*relationship* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols). $\mathcal{E}$ is further partitioned into: a set $\mathcal{E}^S$ of *snapshot entities* (the S-*marked* entities in Figure 2), a set $\mathcal{E}^I$ of I*mplicitly temporal entities* (the *unmarked* entities in Figure 2), and a set $\mathcal{E}^T$ of *temporary entities* (the VT-*marked* entities in Figure 2). A similar partition applies to the set $\mathcal{R}$. ATT is a function that maps an entity symbol in $\mathcal{E}$ to an $\mathcal{A}$-labeled tuple over $\mathcal{D}$, $\text{ATT}(E) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$. REL is a function that maps a relationship symbol in $\mathcal{R}$ to an $\mathcal{U}$-labeled tuple over $\mathcal{E}$, $\text{REL}(R) = \langle U_1 : E_1, \ldots, U_k : E_k \rangle$, and $k$ is the *arity* of $R$. CARD is a function $\mathcal{E} \times \mathcal{R} \times \mathcal{U} \mapsto \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ denoting cardinality constraints. We denote with $\text{CMIN}(E, R, U)$ and $\text{CMAX}(E, R, U)$ the first and second component of CARD. In Figure 2, $\text{CARD}(\texttt{TopManager}, \texttt{Manages}, \texttt{man}) = (1, 1)$. ISA is a binary relationship $\text{ISA} \subseteq (\mathcal{E} \times \mathcal{E}) \cup (\mathcal{R} \times \mathcal{R})$. ISA between relationships is restricted to relationships with the same

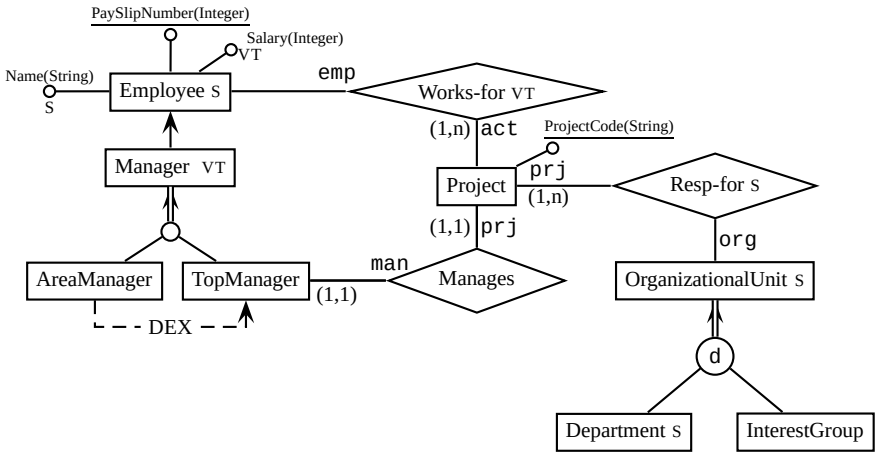



Figure 2: An $\mathcal{ER}_{VT}$ diagram

arity. ISA is visualized with a directed arrow, e.g. `Manager` ISA `Employee` in Figure 2. DISJ, COVER are binary relations over $2^{\mathcal{E}} \times \mathcal{E}$, describing disjointness and covering partitions, respectively. DISJ is visualized with a circled "d" and COVER with a double directed arrow, e.g. `Department`, `InterestGroup` are both disjoint and they cover `OrganizationalUnit`. S, T are binary relations over $\mathcal{E} \times \mathcal{A}$ containing, respectively, the snapshot and temporary attributes of an entity (see S, T marked attributes in Figure 2). KEY is a function that maps entity symbols in $\mathcal{E}$ to their key attributes, $\text{KEY}(E) = A$. Keys are visualized as underlined attributes. Both DEX and PER are binary relations over $\mathcal{E} \times \mathcal{E}$ describing the dynamic evolution of entities. DEX and PER are visualized with dotted directed lines labeled with DEX or PER, respectively (e.g. `AreaManager` DEX `TopManager`).

The model-theoretic semantics associated with the $\mathcal{ER}_{VT}$ modeling language adopts the *snapshot*[2] representation of abstract temporal databases and temporal conceptual models [8]. Following this paradigm, the flow of time $\mathcal{T} = \langle \mathcal{T}_p, < \rangle$, where $\mathcal{T}_p$ is a set of time points (or chronons) and $<$ is a binary precedence relation on $\mathcal{T}_p$, is assumed to be isomorphic to either $\langle \mathbb{Z}, < \rangle$ or $\langle \mathbb{N}, < \rangle$. Thus, a temporal database can be regarded as a mapping from time points in $\mathcal{T}$ to standard relational databases, with the same interpretation of constants and the same domain.

**Definition 3.1 ($\mathcal{ER}_{VT}$ Semantics).** *Let $\Sigma$ be an $\mathcal{ER}_{VT}$ schema. A* temporal database state *for the schema $\Sigma$ is a tuple $\mathcal{B} = (\mathcal{T}, \Delta^{\mathcal{B}} \cup \Delta^{\mathcal{B}}_D, \cdot^{\mathcal{B}(t)})$, such that: $\Delta^{\mathcal{B}}$ is a nonempty set disjoint from $\Delta^{\mathcal{B}}_D$; $\Delta^{\mathcal{B}}_D = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of basic domain values used in the schema $\Sigma$; $\cdot^{\mathcal{B}(t)}$ is a function such that for each $t \in \mathcal{T}$, every domain symbol $D_i \in \mathcal{D}$, every entity $E \in \mathcal{E}$, every relationship $R \in \mathcal{R}$, and every attribute $A \in \mathcal{A}$, we have: $D_i^{\mathcal{B}(t)} = \Delta^{\mathcal{B}}_{D_i}$, $E^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}}$, $R^{\mathcal{B}(t)}$ is a set of $\mathcal{U}$-labeled tuples over $\Delta^{\mathcal{B}}$, and $A^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}} \times \Delta^{\mathcal{B}}_D$. $\mathcal{B}$ is a* legal temporal database state *if it satisfies all integrity constraints expressed in the schema. In particular, the interpretation of* ISA, ATT, REL, CARD, DISJ.COVER *is similar to the atemporal case (see [7, 4]). For the temporal constructs we have:*

[2]The snapshot model represents the same class of temporal databases as the *timestamp* model [12, 13] defined by adding temporal attributes to a relation [8].

*For each snapshot entity* $E \in \mathcal{E}^S$*, if,* $e \in E^{\mathcal{B}(t)}$*, then,* $\forall t' \in \mathcal{T}.e \in E^{\mathcal{B}(t')}$.

*For each temporary entity* $E \in \mathcal{E}^T$*, if,* $e \in E^{\mathcal{B}(t)}$*, then,* $\exists t' \neq t.e \notin E^{\mathcal{B}(t')}$.

*For each snapshot relationship* $R \in \mathcal{R}^S$*, if,* $r \in R^{\mathcal{B}(t)}$*, then,* $\forall t' \in \mathcal{T}.r \in R^{\mathcal{B}(t')}$.

*For each temporary relationship* $R \in \mathcal{R}^T$*, if,* $r \in R^{\mathcal{B}(t)}$*, then,* $\exists t' \neq t.r \notin R^{\mathcal{B}(t')}$.

*For each entity* $E \in \mathcal{E}$ *with a snapshot attribute* $A_i$*, i.e.* $\langle E, A_i \rangle \in$ S*, if,* $(e \in E^{\mathcal{B}(t)} \wedge \langle e, a_i \rangle \in A_i^{\mathcal{B}(t)})$*, then,* $\forall t' \in \mathcal{T}.\langle e, a_i \rangle \in A_i^{\mathcal{B}(t')}$.

*For each entity* $E \in \mathcal{E}$ *with a temporary attribute* $A_i$*, i.e.* $\langle E, A_i \rangle \in$ T*, if,* $(e \in E^{\mathcal{B}(t)} \wedge \langle e, a_i \rangle \in A_i^{\mathcal{B}(t)})$*, then,* $\exists t' \neq t.\langle e, a_i \rangle \notin A_i^{\mathcal{B}(t')}$.

*For each* $E \in \mathcal{E}, A \in \mathcal{A}$ *such that* $\text{KEY}(E) = A$*, then,* $\langle E, A_i \rangle \in$ S*—i.e. a key is a snapshot attribute—and* $\forall a \in \Delta_D^{\mathcal{B}}.\#\{e \in E^{\mathcal{B}(t)} \mid \langle e, a \rangle \in A^{\mathcal{B}(t)}\} \leq 1$.

*For each* $E_1, E_2 \in \mathcal{E}$*, if* $E_1$ DEX $E_2$*, if,* $e \in E_1^{\mathcal{B}(t)}$*, then,* $\exists t_1 > t.e \in E_2^{\mathcal{B}(t_1)}$*;*

*For each* $E_1, E_2 \in \mathcal{E}$*, if* $E_1$ PER $E_2$*, if,* $e \in E_1^{\mathcal{B}(t)}$*, then,* $\forall t' > t.e \in E_2^{\mathcal{B}(t')}$.

# 4 Reasoning on Temporal Models

Reasoning tasks over a temporal conceptual model include verifying whether an entity, relationship, or schema are *satisfiable*, whether a *subsumption* relation exists between entities or relationships, or checking whether a new schema property is *logically implied* by a given schema. The model-theoretic semantics associated with $\mathcal{ER}_{VT}$ allows us to formally define these reasoning tasks.

**Definition 4.1 (Reasoning in $\mathcal{ER}_{VT}$).** *Let* $\Sigma$ *be an* $\mathcal{ER}_{VT}$ *schema,* $E \in \mathcal{E}$ *an entity, and* $R \in \mathcal{R}$ *a relationship. The following are the reasoning tasks over* $\Sigma$*:*

1. $E$ $(R)$ *is* satisfiable *if there exists a legal temporal database state* $\mathcal{B}$ *for* $\Sigma$ *such that* $E^{\mathcal{B}(t)} \neq \emptyset$ $(R^{\mathcal{B}(t)} \neq \emptyset)$*, for some* $t \in \mathcal{T}$*;*

2. $E$ $(R)$ *is* liveness satisfiable *if there exists a legal temporal database state* $\mathcal{B}$ *for* $\Sigma$ *such that* $\forall t \in \mathcal{T}.\exists t' > t.E^{\mathcal{B}(t')} \neq \emptyset$ $(R^{\mathcal{B}(t')} \neq \emptyset)$*, i.e.* $E$ $(R)$ *is satisfiable infinitely often;*

3. $E$ $(R)$ *is* globally satisfiable *if there exists a legal temporal database state* $\mathcal{B}$ *for* $\Sigma$ *such that* $E^{\mathcal{B}(t)} \neq \emptyset$ $(R^{\mathcal{B}(t)} \neq \emptyset)$*, for all* $t \in \mathcal{T}$*;*

4. $\Sigma$ *is* satisfiable *if there exists a legal temporal database state* $\mathcal{B}$ *for* $\Sigma$ *that satisfies at least one entity in* $\Sigma$ *(*$\mathcal{B}$ *is said a* model *for* $\Sigma$*);*

5. $E_1$ $(R_1)$ *is* subsumed *by* $E_2$ $(R_2)$ *in* $\Sigma$ *if every legal temporal database state for* $\Sigma$ *is also a legal temporal database state for* $E_1$ ISA $E_2$ $(R_1$ ISA $R_2)$*;*

6. *A schema* $\Sigma'$ *is* logically implied *by a schema* $\Sigma$ *over the same signature if every legal temporal database state for* $\Sigma$ *is also a legal temporal database state for* $\Sigma'$.

Based on this formal characterization the following Proposition proves that reasoning services (1-5) relative to entities are mutually reducible to each other. As far as relationships are concerned, the reasoning services (1-3) can be reduced to analogous problems for entities.

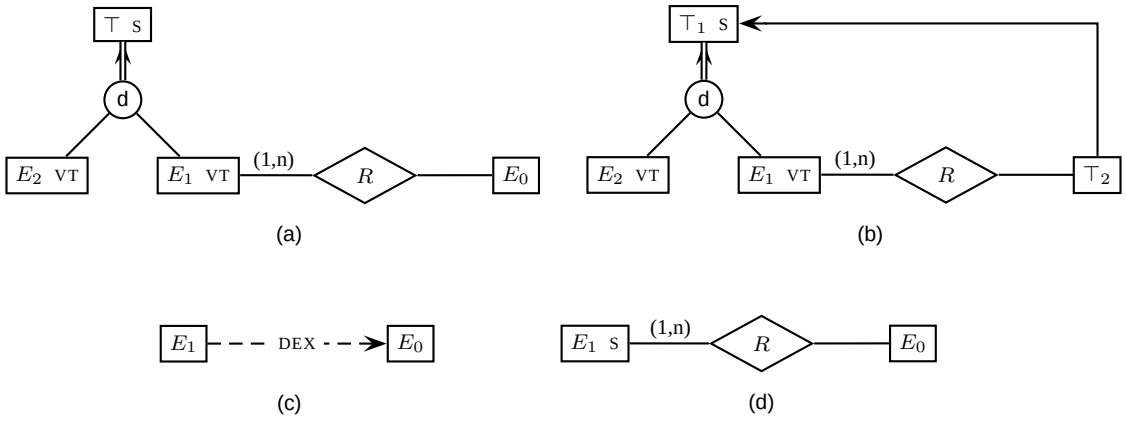



Figure 3: Reductions: (a) From Entity Sat to Schema Sat; (b) From Schema Sat to Entity Liveness Sat; (c) From Entity Liveness Sat to Entity Global Sat; (d) From Entity Global Sat to Entity Sat.

Indeed, we can verify whether a relationship $R$ is satisfiable in $\Sigma$ by adding a new entity, say $A_R$ such that: (a) $A_R$ ISA $E$, with $E$ an arbitrary entity participating in the relationship, and (b) $A_R$ totally participates in the relationship. Then, $R$ is satisfiable (liveness or globally satisfiable) if and only if $A_R$ is satisfiable (liveness or globally satisfiable). As far as relationships subsumption is concerned, it can be reduced to relationships satisfiability by extending $\mathcal{ER}_{VT}$ to express disjoint hierarchies between relationships and then applying the reduction proposed by [6] for entities.

**Proposition 4.2.** *There is a mutual reducibility between the reasoning services (1-5) in $\mathcal{ER}_{VT}$.*

*Proof.* (`Sketch.`)

1. Proving the mutual reducibility between satisfiability and subsumption in $\mathcal{ER}_{VT}$ can be done similarly to [6].

2. Entity satisfiability reduces to schema satisfiability.
   An arbitrary entity, $E_0$, is satisfiable w.r.t. $\Sigma$ iff a new schema $\Sigma'$ is satisfiable. $\Sigma'$ is obtained by adding to $\Sigma$ the schema in Figure 3(a), where $\top, E_1, E_2$ are new entities such that $\forall E \in \mathcal{E}.E$ ISA $\top$, and $R$ is a new binary relationship.

3. Schema satisfiability reduces to entity liveness satisfiability.
   An arbitrary schema $\Sigma$ is satisfiable iff an entity is liveness satisfiable w.r.t. a new schema $\Sigma'$. $\Sigma'$ is obtained by adding to $\Sigma$ the schema in Figure 3(b), where $\top_1, \top_2, E_1, E_2$ are new entities and $R$ is a new binary relationship. Furthermore, $\{E \mid E \in \mathcal{E}\}$ COVER $\top_2$. In particular, $\Sigma$ is satisfiable iff $\top_1$ is liveness satisfiable w.r.t. $\Sigma'$.

4. Entity liveness satisfiability reduces to entity global satisfiability.
   An arbitrary entity, $E_0$, is liveness satisfiable w.r.t. $\Sigma$ iff an entity is globally satisfiable w.r.t. a new schema $\Sigma'$. $\Sigma'$ is obtained by adding to $\Sigma$ the new entity $E_1$ as shown in Figure 3(c). In particular, $E_0$ is liveness satisfiable w.r.t. $\Sigma$ iff $E_1$ is globally satisfiable w.r.t. $\Sigma'$.

5. Entity global satisfiability reduces to entity satisfiability.
   An arbitrary entity, $E_0$, is globally satisfiable w.r.t. $\Sigma$ iff the new entity $E_1$ is satisfiable w.r.t. the new schema $\Sigma'$. $\Sigma'$ is obtained by adding to $\Sigma$ the schema in Figure 3(d), where $E_1$ is new snapshot entity and $R$ is a new binary relationship. □

Finally, we show that all the reasoning problems can be reduced to a logical implication problem. Indeed, checking whether an entity $E$ is satisfiable can be reduced to logical implication by choosing $\Sigma' = \{E \text{ ISA } A, E \text{ ISA } B, \{A, B\} \text{ DISJ } C\}$, with $A, B, C$ arbitrary entities. Then, $E$ is satisfiable iff $\Sigma \not\models \Sigma'$. Given the result of Proposition 4.2, then the reasoning services (1-5) for entities are reducible to logical implication. Furthermore, given two relationships $R_1, R_2$, checking for sub-relationship can be reduced to logical implication by choosing $\Sigma' = \{R_1 \text{ ISA } R_2\}$.

# 5 Reasoning on $\mathcal{ER}_{VT}$ is Undecidable

We now show that reasoning on full $\mathcal{ER}_{VT}$ is undecidable. The proof is based on a reduction from the undecidable halting problem for a Turing machine to the entity satisfiability problem w.r.t. an $\mathcal{ER}_{VT}$ schema $\Sigma$. We apply ideas similar to [9] (Sect. 7.5) to show undecidability of certain products of modal logics. The proof can be divided in the following two steps: 1. Reduction of the halting problem to concept satisfiability w.r.t. an $\mathcal{ALC}_{\mathsf{F}}$ KB; 2. Reduction of concept satisfiability w.r.t. an $\mathcal{ALC}_{\mathsf{F}}$ KB to entity satisfiability w.r.t. an $\mathcal{ER}_{VT}$ schema.

### Reasoning on $\mathcal{ALC}_{\mathsf{F}}$ is undecidable

Using a reduction from the halting problem we now prove that reasoning involving an $\mathcal{ALC}_{\mathsf{F}}$ knowledge base is undecidable. In [9] the undecidability of $\mathcal{ALC}_{\mathsf{F}}$ is proved using: (a) complex axioms—i.e. axioms can be combined using Boolean and modal operators—(b) both *global* and *local* axioms—i.e. axioms can be either true at all time or true at some time, respectively. Since $\mathcal{ER}_{VT}$ is able to encode just simple global axioms, we modify the proof presented in [9].

**Proposition 5.1.** *Concept satisfiability w.r.t. an $\mathcal{ALC}_{\mathsf{F}}$ KB is undecidable.*

*Proof.* (`Sketch.`) A single-tape right-infinite deterministic Turing machine, $\mathbf{M}$, is a triple $\langle A, S, \rho \rangle$, where: $A$ is the *tape alphabet* ($b \in A$ stands for blank); $S$ is a finite set of *states* with *initial state*, $s_0$, and *final state*, $s_1$; $\rho$ is the *transition function*, $\rho : (S - \{s_1\}) \times A \to S \times (A \cup \{\mathtt{L}, \mathtt{R}\})$. We construct an $\mathcal{ALC}_{\mathsf{F}}$ KB, say $\mathbf{KB}_M$, with a concept that is satisfiable w.r.t. $\mathbf{KB}_M$ iff the machine $\mathbf{M}$ does not halt. We introduce some shortcuts. The implication, $C \to D$, is equivalent to $\neg C \sqcup D$. We define $\mathtt{next}(C, D)$ as: $C \sqsubseteq \Diamond^+ D \sqcap \neg\Diamond^+\Diamond^+ D$. Finally, $\mathtt{discover}(C, \{D_1, \ldots, D_n\})$ is the disjoint covering between $C$ and $D_1 \ldots D_n$. Let $A' = A \cup \{\pounds\} \cup (S \times A)$, where $\pounds \notin A$ is a symbol marking the left end of the tape. With each $x \in A'$ we introduce a concept $C_x$. We also use concepts $C_s, C_l, C_r$ to denote the active cell, its left and right cells, respectively. The concept $S1$ denotes the final state. The halting problem reduces to satisfiability of $C_0$. Extra concepts $C, D_1, D_2, D_3$, will be also used. $R$ is a global role. $\mathbf{KB}_M$ contains the following axioms:

$$C_0 \sqsubseteq C_{\pounds} \sqcap \Diamond^+ C_{\langle s_0,b\rangle} \quad (1)$$
$$\top \sqsubseteq \exists R.\top \quad (2)$$
$$\texttt{next}(C_{\pounds}, D_1) \quad (3)$$
$$\texttt{next}(D_1, D_2) \quad (4)$$
$$C_{\langle s_0,b\rangle} \sqsubseteq D_1 \quad (5)$$
$$C_{\langle s_0,b\rangle} \sqsubseteq \Box^+ C_b \quad (6)$$
$$\texttt{next}(C_l, C_s) \quad (7)$$
$$\texttt{next}(C_s, C_r) \quad (8)$$
$$\texttt{next}(C_r, D_3) \quad (9)$$
$$C_{\pounds} \sqsubseteq C_l \sqcup \Diamond^+ C_l \quad (10)$$
$$C_l \sqsubseteq C_\alpha \rightarrow \forall R.C_{\alpha'} \quad (11)$$
$$C_s \sqsubseteq C_\beta \rightarrow \forall R.C_{\beta'} \quad (12)$$
$$C_r \sqsubseteq C_\gamma \rightarrow \forall R.C_{\gamma'} \quad (13)$$
$$C_a \sqsubseteq (\neg C_l \sqcap \neg C_s \sqcap \neg C_r) \rightarrow \forall R.C_a, \ \ \forall a \in A \cup \{\pounds\} \quad (14)$$
$$\texttt{discover}(S1, \{C_{\langle s_1,a\rangle} \mid a \in A \cup \{\pounds\}\}) \quad (15)$$
$$\texttt{discover}(C, \{C_x \mid x \in A'\}) \quad (16)$$
$$\texttt{discover}(C_s, \{C_{\langle s,a\rangle} \mid \langle s,a\rangle \in S \times A\}) \quad (17)$$
$$C_s \sqsubseteq \neg S1 \quad (18)$$

with axioms (11–13) for each instruction, $\delta(\alpha, \beta, \gamma) = \langle \alpha', \beta', \gamma' \rangle$, defined as

$$\delta(a_i, \langle s, a_j\rangle, a_k) = \begin{cases} \langle a_i, \langle s', a_j'\rangle, a_k\rangle, & \text{if } \rho(s, a_j) = \langle s', a_j'\rangle \\ \langle \langle s', a_i\rangle, a_j, a_k\rangle, & \text{if } \rho(s, a_j) = \langle s', \mathtt{L}\rangle \text{ and } a_i \neq \pounds \\ \langle \pounds, \langle s', a_j\rangle, a_k\rangle, & \text{if } \rho(s, a_j) = \langle s', \mathtt{L}\rangle \text{ and } a_i = \pounds \\ \langle a_i, a_j, \langle s', a_k\rangle\rangle, & \text{if } \rho(s, a_j) = \langle s', \mathtt{R}\rangle \end{cases}$$

We can prove that $C_0$ is satisfiable w.r.t. $\mathbf{KB}_M$ iff $\mathbf{M}$ has an infinite computation starting from the empty tape. □

### Reducing $\mathcal{ALC}_{\mathsf{F}}$ concept sat to $\mathcal{ER}_{VT}$ entity sat

We now show how to capture the $\mathcal{ALC}_{\mathsf{F}}$ knowledge base $\mathbf{KB}_M$ with an $\mathcal{ER}_{VT}$ schema, $\Sigma_M$. The mapping is based on a similar reduction presented in [6] for capturing $\mathcal{ALC}$ axioms. For each atomic concept and role in $\mathbf{KB}_M$ we introduce an entity and a relationship, respectively. To simulate the universal concept, $\top$, we introduce a snapshot entity, `Top`, that generalizes all the entities in $\Sigma_M$. Additionally, the various axioms in $\mathbf{KB}_M$ are encoded in $\mathcal{ER}_{VT}$ as follows:

1. Axioms involving `discover` are mapped using disjoint and covering hierarchies.

2. Axioms of the form $C \sqsubseteq D$, with $C, D$ atomic concepts are encoded as $C$ ISA $D$.

3. For axioms of the form $C \sqsubseteq \neg D$ we construct the hierarchy in Figure 4(a).

4. For axioms of the form $C \sqsubseteq D_1 \sqcup \ldots \sqcup D_n$ we construct the hierarchy in Figure 4(b).

5. Axioms of the form $C \sqsubseteq \forall R.D$ are mapped together with the axiom $\top \sqsubseteq \exists R.\top$ by introducing a new sub-relationship, $R_C$, and considering $R$ as a functional role[3]. Figure 4(c) shows the mapping where $R$ is a snapshot relationship to capture the fact that $R$ is a global role in $\mathbf{KB}_M$.

6. For each axiom of the form $C \sqsubseteq \Box^+ D$ ($C \sqsubseteq \Diamond^+ D$) we use a persistency (respectively, dynamic extension) constraint: $C$ PER $D$ (respectively, $C$ DEX $D$).

7. Axioms of the form $\texttt{next}(C, D)$ are mapped by using the dynamic extension constraint to capture that $C \sqsubseteq \Diamond^+ D$. To capture that $C \sqsubseteq \neg\Diamond^+\Diamond^+ D$ we rewrite it as $C \sqsubseteq \Box^+\Box^+\neg D$, which, in turn, is encoded by the following axioms: $C \sqsubseteq \Box^+ C_1$; $C_1 \sqsubseteq \Box^+ C_2$; $C_2 \sqsubseteq \neg D$. Figure 4(d) shows the diagram that maps `next` axioms.

[3]Considering $R$ as a functional role does not change the $\mathcal{ALC}_{\mathsf{F}}$ undecidability proof.

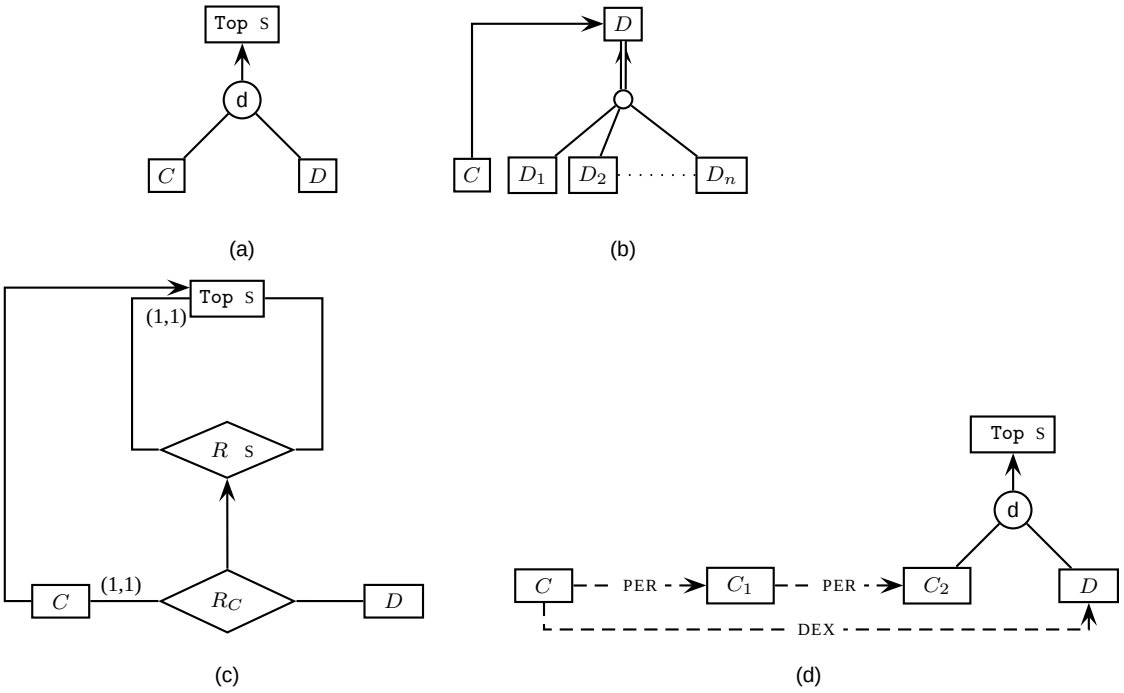



Figure 4: Encoding axioms: (a) $C \sqsubseteq \neg D$; (b) $C \sqsubseteq D_1 \sqcup \ldots \sqcup D_n$; (c) $C \sqsubseteq \forall R.D$ and $\top \sqsubseteq \exists R.\top$; (d) $\texttt{next}(C, D)$.

The above reductions are enough to capture all axioms in $\mathbf{KB}_M$. Indeed, axioms (11–13) have the form: $C \sqsubseteq \neg C_1 \sqcup \forall R.C_2$, while axioms (16) have the form: $C_a \sqsubseteq C_l \sqcup C_s \sqcup C_r \sqcup \forall R.C_a$. We are now able to prove the main result of this paper.

**Theorem 5.2.** *Reasoning in $\mathcal{ER}_{VT}$ using persistency and dynamic constructs is undecidable.*

*Proof.* Proving that the above reduction from $\mathbf{KB}_M$ to $\Sigma_M$ is true can be easily done by checking the semantic equivalence between each $\mathcal{ALC}_{\mathsf{F}}$ axiom and its encoding (for a similar proof see [6]). Then, the concept $C_0$ is satisfiable w.r.t. $\mathbf{KB}_M$ iff the entity $C_0$ is satisfiable w.r.t. $\Sigma_M$. Thus, because of Proposition 5.1, the halting problem can be reduced to reasoning in $\mathcal{ER}_{VT}$. □

**Acknowledgments**

I would like to thank Diego CAlvanese, Enrico Franconi, Sergio Tessaris and Frank Wolter together with the anonymous referees for enlightening comments on earlier drafts of the paper.

# References

[1] A. Artale. Reasoning on temporal conceptual schemas with dynamic constraints. In *Proc. of the 11th Int. Symposium on Temporal Representation and Reasoning, TIME'04*. IEEE Computer Society, July 2004.

[2] A. Artale and E. Franconi. Temporal ER modeling with description logics. In *Proc. of the Int. Conference on Conceptual Modeling (ER'99)*. Springer-Verlag, 1999.

[3] A. Artale and E. Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30("1-4"), 2001.

[4] Alessandro Artale, Enrico Franconi, and Federica Mandreoli. Description logics for modelling dynamic information. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases*. LNCS, Springer, 2003.

[5] Alessandro Artale, Enrico Franconi, Frank Wolter, and Michael Zakharyaschev. A temporal description logic for reasoning about conceptual schemas and queries. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proc. of the 8th Joint European Conference on Logics in Artificial Intelligence (JELIA-02)*, volume 2424 of *LNAI*. Springer, 2002.

[6] Daniela Berardi, Andrea Calì, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. Technical Report 11-03, 2003.

[7] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[8] J. Chomicki and D. Toman. Temporal logic in information systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.

[9] D. Gabbay, A.Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*. Studies in Logic. Elsevier, 2003.

[10] H. Gregersen and J.S. Jensen. Conceptual modeling of time-varying information. Technical Report TimeCenter TR-35, Aalborg University, Denmark, 1998.

[11] C. S. Jensen, J. Clifford, S. K. Gadia, P. Hayes, and S. Jajodia et al. The Consensus Glossary of Temporal Database Concepts. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases - Research and Practice*. Springer-Verlag, 1998.

[12] C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 111(1):36–44, 1999.

[13] C. S. Jensen, M. Soo, and R. T. Snodgrass. Unifying temporal data models via a conceptual model. *Information Systems*, 9(7):513–547, 1994.

[14] S. Spaccapietra, C. Parent, and E. Zimanyi. Modeling time from a conceptual perspective. In *Int. Conf. on Information and Knowledge Management (CIKM98)*, 1998.

[15] C. Theodoulidis, P. Loucopoulos, and B. Wangler. A conceptual modelling formalism for temporal database applications. *Information Systems*, 16(3):401–416, 1991.

[16] F. Wolter and M. Zakharyaschev. Satisfiability problem in description logics with modal operators. In *Proc. of the 6 $^{th}$ International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 512–523, Trento, Italy, June 1998.

# Computing the Least Common Subsumer w.r.t. a Background Terminology*

Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan
Theoretical Computer Science, TU Dresden, Germany
{baader,sertkaya,turhan}@tcs.inf.tu-dresden.de

**Abstract**

Methods for computing the least common subsumer (lcs) are usually restricted to rather inexpressive DLs whereas existing knowledge bases are written in very expressive DLs. In order to allow the user to re-use concepts defined in such terminologies and still support the definition of new concepts by computing the lcs, we extend the notion of the lcs of concept descriptions to the notion of the lcs w.r.t. a background terminology.

## 1 Introduction and problem definition

Non-standard inferences such as computing the least common subsumer can be used to support the *bottom-up* construction of DL knowledge bases, as introduced in [4, 5]: instead of directly defining a new concept, the knowledge engineer introduces several typical examples as objects, which are then automatically generalized into a concept description by the system. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept. The task of computing such a concept description can be split into two subtasks: computing the most specific concepts of the given objects, and then computing the least common subsumer of these concepts. The *most specific concept* (msc) of an object $o$ (the *least common subsumer* (lcs) of concept descriptions $C_1, \ldots, C_n$) is the most specific concept description $C$ expressible in the given DL language that has $o$ as an instance (that subsumes $C_1, \ldots, C_n$). The problem of computing the lcs and (to a more limited extent) the msc has already been investigated in the literature [11, 12, 4, 5, 21, 20, 19, 3, 9].

The methods for computing the least common subsumer are restricted to rather inexpressive descriptions logics not allowing for disjunction (and thus not allowing for full negation). In fact, for languages with disjunction, the lcs of a collection of concepts is just their disjunction, and nothing new can be learned from building it. In contrast, for languages without disjunction, the lcs extracts the "commonalities" of the given collection of concepts. Modern DL systems like FaCT [18] and RACER [17] are based on very expressive DLs, and there exist large knowledge bases that use this

*This work has been supported by the German Research Foundation (DFG) under grants GRK 334/3 and BA 1122/4-3 and National ICT Australia Limited, Canberra Research Lab.

expressive power and can be processed by these systems [22, 23, 16]. In order to allow the user to re-use concepts defined in such existing knowledge bases and still support the user during the definition of new concepts with the bottom-up approach sketched above, we propose the following extended bottom-up approach.

Consider a *background terminology* $\mathcal{T}$ defined in an expressive DL $\mathcal{L}_2$. When defining new concepts, the user employs only a sublanguage $\mathcal{L}_1$ of $\mathcal{L}_2$, for which computing the lcs makes sense. However, in addition to primitive concepts and roles, the concept descriptions written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$. Let us call such concept descriptions $\mathcal{L}_1(\mathcal{T})$-concept descriptions.

**Definition 1** *Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and a collection $C_1, \ldots, C_n$ of $\mathcal{L}_1(\mathcal{T})$-concept descriptions, the* least common subsumer *(lcs) of $C_1, \ldots, C_n$* w.r.t. *$\mathcal{T}$ is the most specific $\mathcal{L}_1(\mathcal{T})$-concept description $C$ that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$, i.e., it is an $\mathcal{L}_1(\mathcal{T})$-concept description $D$ such that*

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

Depending on the DLs $\mathcal{L}_1$ and $\mathcal{L}_2$, least common subsumers of $\mathcal{L}_1(\mathcal{T})$-concept descriptions w.r.t. an $\mathcal{L}_2$-TBox $\mathcal{T}$ may exist or not.

Note that the lcs only uses concept constructors from $\mathcal{L}_1$, but may also contain concept names defined in the $\mathcal{L}_2$-TBox. This is the main distinguishing feature of this new notion of a least common subsumer w.r.t. a background terminology. Let us illustrate this by a small example.

**Example 2** Assume that $\mathcal{L}_1$ is the DL $\mathcal{EL}$ (allowing for conjunction, existential restrictions, and the top concept) and $\mathcal{L}_2$ is $\mathcal{ALC}$ (extending $\mathcal{EL}$ by negation, disjunction, and value restrictions). Consider the $\mathcal{ALC}$-TBox

$$\mathcal{T} := \{A \equiv P \sqcup Q\},$$

and assume that we want to compute the lcs of the $\mathcal{EL}(\mathcal{T})$-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 $\mathcal{EL}$ would be the top concept $\top$.

In the following we always assume that DLs $\mathcal{L}_1$ and $\mathcal{L}_2$ and an $\mathcal{L}_2$-TBox are given, and if we talk about (least) common subsumers we mean the ones in $\mathcal{L}_1(\mathcal{T})$, and not in $\mathcal{L}_1$ or $\mathcal{L}_2$. In the next section, we consider the case where $\mathcal{L}_1$ is $\mathcal{EL}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$ in more detail. We show the following two results:

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

At first sight, one might assume that the first result can be shown using results on approximation of DLs [10]. In fact, given an acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$ and $\mathcal{EL}(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can first unfold $C_1, \ldots, C_n$ into $\mathcal{ALC}$-concept descriptions $C_1', \ldots, C_n'$, then build the $\mathcal{ALC}$-concept description $C := C_1' \sqcup \ldots \sqcup C_n'$, and finally approximate $C$ from above by an $\mathcal{EL}$-concept description $E$. However, $E$ then does not contain concept names defined in $\mathcal{T}$, and thus it is not necessarily the least $\mathcal{EL}(\mathcal{T})$-concept description subsuming $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$ (see Example 2 above). One might now assume that this can be overcome by applying known results on rewriting concept descriptions w.r.t. a terminology [6]. However, in Example 2, the concept description $E$ obtained using the approach based on approximation sketched above is $\top$, and this concept cannot be rewritten using the TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$.

The result on the existence and computability of the lcs w.r.t. a background terminology shown in the next section is theoretical in the sense that it does not yield a practical algorithm. In Section 3 we follow a more practical approach. Assume that $\mathcal{L}_1$ is a DL for which least common subsumers (without background TBox) always exist. Given $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can compute a common subsumer w.r.t. $\mathcal{T}$ by just ignoring $\mathcal{T}$, i.e., by treating the defined names in $C_1, \ldots, C_n$ as primitive and computing the lcs of $C_1, \ldots, C_n$ in $\mathcal{L}_1$. In Section 3 we sketch practical methods for computing "good" common subsumers w.r.t. background TBoxes, which may not be the *least* common subsumers, but which are better than the common subsumers computed by just ignoring the TBox.

## 2 Two exact theoretical results

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 3** *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 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 $k$.

To show that these facts imply Theorem 3 consider the $\mathcal{EL}(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$. If all of them are unsatisfiable w.r.t. $\mathcal{T}$, then one of them (e.g., $C_1$) can be taken as their lcs w.r.t. $\mathcal{T}$. Otherwise, assume that $C_i$ is satisfiable w.r.t. $\mathcal{T}$. Let $C_i'$ be the $\mathcal{ALC}$-concept description obtained by unfolding $C_i$ w.r.t. $\mathcal{T}$, and assume that its role depth is $\ell$. Now, take an arbitrary $\mathcal{EL}(\mathcal{T})$-concept description $E$ that is a common subsumer of $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. Then, the role depth of $E$ is at most $\ell$. Otherwise, $C_i \sqsubseteq_{\mathcal{T}} E$ would be in contradiction to the above facts 1. and 2. Thus, fact 3. implies that, up to equivalence, there are only finitely many common subsumers of $C_1, \ldots, C_n$ in $\mathcal{EL}(\mathcal{T})$. The least common subsumer is simply the conjunction of these finitely many $\mathcal{EL}(\mathcal{T})$-concept descriptions.

It is not hard to see that the above proof is effective in the sense that one can effectively compute (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 4** *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.*

*Proof.* Let $E_n$ denote the $\mathcal{EL}$-concept description $\exists r.\exists r. \ldots \exists r.\top$ of role depth $n$. For all $n \geq 0$, $E_n$ is a common subsumer of $A$ and $B$ w.r.t. $\mathcal{T}$. Assume that $D$ is a least common subsumer of $A$ and $B$, and let $\ell$ be the role depth of $D$. If $D$ contains neither $A$ nor $B$, then $D \not\sqsubseteq_{\mathcal{T}} E_n$ for all $n > \ell$, which is a contradiction. However, if $D$ contains $A$, then it is easy to see that $D$ cannot be a subsumer of $B$, and if $D$ contains $B$, then it cannot be a subsumer of $A$. Consequently, such a least common subsumer $D$ cannot exist. □

Note that this example is very similar to the one showing non-existence of the lcs in $\mathcal{EL}$ with cyclic terminologies interpreted with descriptive semantics [2]. However, the proof of the result in [2] is more complicated since there one is allowed to extend the terminology in order to build the lcs.

## 3 A practical approximative approach

We have seen above that the lcs w.r.t. general background TBoxes need not exist. In addition, even in the case of acyclic TBoxes, where the lcs always exists, we do not have a practical algorithm for computing the lcs. In the bottom-up construction of DL knowledge bases, it is not really necessary to use the *least* common subsumer,[1] 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 $\mathcal{EL}$-concept descriptions can be computed.

[1] Using it may even result in over-fitting.

### The lcs of $\mathcal{EL}$-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 lcs $\mathsf{lcs}_{\mathcal{EL}}(C, D)$ of two $\mathcal{EL}$-concept descriptions $C, D$.

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

Now, let $C, D$ be $\mathcal{EL}$-concept descriptions. Then we have

$$\mathsf{lcs}_{\mathcal{EL}}(C, D) \quad = \quad \bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \ \sqcap \bigsqcap_{r \in \mathsf{roles}(C) \cap \mathsf{roles}(D)} \ \bigsqcap_{E \in \mathsf{restrict}_r(C), F \in \mathsf{restrict}_r(D)} \exists r.\mathsf{lcs}_{\mathcal{EL}}(E, F)$$

Here, the empty conjunction stands for the top concept $\top$. The recursive call of $\mathsf{lcs}_{\mathcal{EL}}$ is well-founded since the role depth of the concept descriptions in $\mathsf{restrict}_r(C)$ ($\mathsf{restrict}_r(D)$) is strictly smaller than the role depth of $C$ ($D$).

### A good common subsumer in $\mathcal{EL}$ w.r.t. a background TBox

Let $\mathcal{T}$ be a background TBox (acyclic or general) in some DL $\mathcal{L}_2$ extending $\mathcal{EL}$ such that subsumption in $\mathcal{L}_2$ w.r.t. this class of TBoxes is decidable. Let $C, D$ be $\mathcal{EL}(\mathcal{T})$-concept descriptions. If we ignore the TBox, then we can simply apply the above algorithm for $\mathcal{EL}$-concept descriptions to compute a common subsumer. However, in this context taking

$$\bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A$$

is not the best we can do. In fact, some of these concept names may be constrained by the TBox, and thus there may be relationships between them that we ignore by simply using the intersection.

Instead, we propose to take the smallest (w.r.t. subsumption w.r.t. $\mathcal{T}$) conjunction of concept names that subsumes (w.r.t. $\mathcal{T}$) both

$$\bigsqcap_{A \in \mathsf{names}(C)} A \qquad \text{and} \qquad \bigsqcap_{B \in \mathsf{names}(D)} B.$$

We modify the above lcs algorithm in this way, not only on the top level of the input concepts, but also in the recursive steps. It is easy to show that the $\mathcal{EL}(\mathcal{T})$-concept description computed by this modified algorithm still is a common subsumer of $A, B$ w.r.t. $\mathcal{T}$. In general, this common subsumer will be more specific than the one obtained by ignoring $\mathcal{T}$, though it need not be the least common subsumer.

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

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

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

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

If we ignore the TBox, then we obtain the $\mathcal{EL}$-concept description $\exists \mathsf{has\text{-}child}.\top$ as common subsumer of $C, D$. However, if we take into account that both $\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}$ and $\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}$ are subsumed by the concept $\mathsf{ChildrenDoctor}$, then we obtain the more specific common subsumer

$$\exists \mathsf{has\text{-}child}.\mathsf{ChildrenDoctor}.$$

**Computing the subsumption lattice of conjunctions of concept names**

In order to obtain a practical lcs algorithm realizing the approach described above, we must be able to compute in an efficient way the smallest conjunction of concept names that subsumes two such conjunctions w.r.t. $\mathcal{T}$. We propose to precompute this information using methods from formal concept analysis (FCA) [15]. In FCA, the knowledge about an application domain is given by means of a formal context.

**Definition 5** *A* formal context *is a triple $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$, where $\mathcal{O}$ is a set of objects, $\mathcal{P}$ is a set of attributes (or properties), and $\mathcal{S} \subseteq \mathcal{O} \times \mathcal{P}$ is a relation that connects each object $o$ with the attributes satisfied by $o$.*

Let $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ be a formal context. For a set of objects $A \subseteq \mathcal{O}$, $A'$ is the set of attributes that are satisfied by all objects in $A$, i.e.,

$$A' := \{p \in \mathcal{P} \mid \forall a \in A\colon (a, p) \in \mathcal{S}\}.$$

Similarly, for a set of attributes $B \subseteq \mathcal{P}$, $B'$ is the set of objects that satisfy all attributes in $B$, i.e.,

$$B' := \{o \in \mathcal{O} \mid \forall b \in B\colon (o, b) \in \mathcal{S}\}.$$

A *formal concept* is a pair $(A, B)$ consisting of an *extent* $A \subseteq \mathcal{O}$ and an *intent* $B \subseteq \mathcal{P}$ such that $A' = B$ and $B' = A$. Such formal concepts can be hierarchically ordered by inclusion of their extents, and this order (denoted by $\leq$ in the following) induces a complete lattice, called the *concept lattice* of the context. Given a formal context, the first step for analyzing this context is usually to compute the concept lattice.

In many applications, one has a large (or even infinite) set of objects, but only a relatively small set of attributes. Also, the context is not necessarily given explicitly as a cross table; it is rather "known" to a domain "expert". In such a situation, Ganter's *attribute exploration* algorithm [13, 15] has turned out to be an efficient approach for computing an appropriate representation of the concept lattice. This algorithm is interactive in the sense that at certain stages it asks the "expert" certain questions about the context, and then continues using the answers provided by the expert. Once the representation of the concept lattice is computed, certain questions about the lattice (e.g. "What is the supremum of two given concepts?") can efficiently be answered using this representation.

Recall that we are interested in the subsumption lattice[2] of conjunctions of concept names (some of which may occur in GCIs or concept definitions of an $\mathcal{L}_2$-TBox $\mathcal{T}$). In order to apply attribute exploration to this task, we define a formal context whose concept lattice is isomorphic to the subsumption lattice we are interested in. This problem was first addressed in [1], where the objects of the context were basically all possible counterexamples to subsumption relationships, i.e., interpretations together with an element of the interpretation domain. The resulting "semantic context" has the disadvantage that an "expert" for this context must be able to deliver such counterexample, i.e., it is not sufficient to have a simple subsumption algorithm for the DL in question. One needs one that, given a subsumption problem "$C \sqsubseteq D$?", is able to compute a counterexample if the subsumption relationship does not hold, i.e., an interpretation $\mathcal{I}$ and an element $d$ of its domain such that $d \in C^{\mathcal{I}} \setminus D^{\mathcal{I}}$.

To overcome this problem, a new "syntactic context" was recently defined in [8]:

**Definition 6** *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 is shown in [8]:

**Theorem 7** *(1) 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. $\mathcal{T}$.*
*(2) Any decision procedure for subsumption w.r.t. TBoxes in $\mathcal{L}_2$ functions as an expert for the context $\mathcal{K}_{\mathcal{T}}$.*

It should be noted that formal concept analysis and attribute exploration has already been applied in a different context to the problem of computing the least common subsumer. In [7], the following problem is addressed: given a finite collection $\mathcal{C}$ of concept descriptions, compute the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$. Again, this extended subsumption hierarchy can be computed by defining a formal context whose concept lattice is isomorphic to the subsumption

[2] In general, the subsumption relation induces a partial order, and not a lattice structure on concepts. However, in the case of conjunctions of concept names, all infima exist, and thus also all suprema.

lattice we are interested in, and then applying attribute exploration (see [7] for details). In [8], it is shown that this approach and the one sketched above can be seen as two instances of a more abstract approach.

**Extension to DLs more expressive than $\mathcal{EL}$**

For the DL $\mathcal{ALE}$ (which extends $\mathcal{EL}$ by value restrictions and atomic negation), an lcs algorithm similar to the one described for $\mathcal{EL}$ exists [5]. The main differences are that (i) the concept descriptions must first be normalized (which may lead to an exponential blow-up); (ii) the recursive calls also deal with value restrictions, and not just existential restrictions; and (iii) on the top level, one has to deal with a conjunction of concept names and *negated* concept names. In the lcs algorithm, the conjunctions mentioned in (iii) are treated similarly to the case of $\mathcal{EL}$ (unless they are contradictory): one separately computes the intersections of the positive and of the negative concept names.

When adapting this algorithm to one that computes "good" common subsumers in $\mathcal{ALE}$ w.r.t. a background TBox, all we have to change is to compute a conjunction of concept names and negated concept names that is the most specific such conjunction subsuming the given conjunctions w.r.t. the TBox, rather than building the intersections. It is easy to see that attribute exploration can again be used to precompute the necessary information. Basically, the only change is that now both concept names and negated concept names are attributes in the formal context.

## 4 Future work

The attributes of the formal contexts introduced in our approach (concept names and possibly negated concept names) are not independent of each other. For example, the name $A$ and its negation $\neg A$ are disjoint, i.e., it is not possible for an object (other than $\bot$) of the context to satisfy both $A$ and $\neg A$. In addition, the TBox induces subsumption relationships between the attributes (and this information may already be precomputed for the given TBox during classification). Thus, one can try to apply a modified version of attribute exploration that can use such background knowledge [14] to speed up the exploration process.

## References


[1] Franz Baader. Computing a minimal representation of the subsumption lattice of all conjunctions of concepts defined in a terminology. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proc. of the 1st Int. KRUSE Symposium*, pages 168–178, 1995.

[2] Franz 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.

[3] Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 319–324. Morgan Kaufmann, 2003.

[4] Franz Baader and Ralf Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proc. of the 22th German Annual Conf. on Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1998.

[5] Franz Baader, Ralf Küsters, and Ralf 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.

[6] Franz Baader, Ralf Küsters, and Ralf Molitor. Rewriting concepts using terminologies. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-00)*, pages 297–308, 2000.

[7] Franz Baader and Ralf Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In B. Ganter and G. Mineau, editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues – Proceedings of the 8th International Conference on Conceptual Structures (ICCS2000)*, volume 1867 of *Lecture Notes In Artificial Intelligence*, pages 290–303. Springer-Verlag, 2000.

[8] Franz Baader and Baris 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.

[9] 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, *Proceedings of the tenth International Conference on Logic for Programming and Automated Reasoning (LPAR'03)*, LNCS. Springer, 2003.

[10] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin 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.

[11] William W. Cohen and Haym Hirsh. Learning the CLASSIC description logics: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 121–133, 1994.

[12] Michael Frazier and Leonard Pitt. CLASSIC learning. *Machine Learning*, 25:151–193, 1996.

[13] Bernhard Ganter. Finding all closed sets: A general approach. *Order*, 8:283–290, 1991.

[14] Bernhard Ganter. Attribute exploration with background knowledge. *Theoretical Computer Science*, 217(2):215–233, 1999.

[15] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, 1999.

[16] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI-01)*, 2001.

[17] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR-01)*, 2001.

[18] Ian 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.

[19] Ralf Küsters and Alex Borgida. What's in an attribute? Consequences for the least common subsumer. *Journal of Artificial Intelligence Research*, 14:167–203, 2001.

[20] Ralf Küsters and Ralf Molitor. Approximating most specific concepts in description logics with existential restrictions. In Franz Baader, Gerd Brewka, and Thomas Eiter, editors, *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI 2001)*, volume 2174 of *Lecture Notes In Artificial Intelligence*, pages 33–47, Vienna, Austria, 2001. Springer-Verlag.

[21] Ralf Küsters and Ralf Molitor. Computing least common subsumers in $\mathcal{ALEN}$. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI-01)*, pages 219–224, 2001.

[22] Alan Rector and Ian 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.

[23] Stefan Schultz and Udo Hahn. Knowledge engineering by large-scale knowledge reuse—experience from the medical domain. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-00)*, pages 601–610. Morgan Kaufmann, 2000.

# On Subsumption and Instance Problem in $\mathcal{ELH}$ w.r.t. General TBoxes

Sebastian Brandt*
Institut für Theoretische Informatik
TU Dresden, Germany
brandt@tcs.inf.tu-dresden.de

**Abstract**

Recently, it was shown for the DL $\mathcal{EL}$ that subsumption and instance problem w.r.t. cyclic terminologies can be decided in polynomial time. In this paper, we show that both problems remain tractable even when admitting general concept inclusion axioms and simple role inclusion axioms.

## 1 Motivation

In the area of DL based knowledge representation, the utility of *general* TBoxes, i.e., TBoxes that allow for general concept inclusion (GCI) axioms, is well known. For instance, in the context of the medical terminology Galen [18], GCIs are used especially for two purposes [16]:

- indicate the status of objects: instead of introducing several concepts for the same concept in different states, e.g., normal insulin secretion, abnormal but harmless insulin secretion, and pathological insulin secretion, only insulin secretion is defined while the status, i.e., normal, abnormal but harmless, and pathological, is implied by GCIs of the form $\ldots \sqsubseteq \exists$has_status.pathological.
- to bridge levels of granularity and add implied meaning to concepts. A classical example [11] is to use a GCI like

$$\begin{aligned}&\mathsf{ulcer} \sqcap \exists \mathsf{has\_loc.stomach}\\ &\quad \sqsubseteq \mathsf{ulcer} \sqcap \exists \mathsf{has\_loc.}(\mathsf{lining} \sqcap \exists \mathsf{is\_part\_of.stomach})\end{aligned}$$

  to render the description of 'ulcer of stomach' more precisely to 'ulcer of lining of stomach' if it is known that 'ulcer of stomach' is specific of the lining of the stomach.

It has been argued that the use of GCIs facilitates the re-use of data in applications of different levels of detail while retaining all inferences obtained from the full description [18]. Hence, to examine reasoning w.r.t. general TBoxes has a strong practical motivation.

*Supported by the DFG under Grant BA 1122/4-3

Research on reasoning w.r.t. general TBoxes has mainly focused on very expressive DLs, reaching as far as, e.g., $\mathcal{ALCNR}$ [5] and $\mathcal{SHIQ}$ [12], in which deciding subsumption of concepts w.r.t. general TBoxes is EXPTIME hard. Fewer results exist on subsumption w.r.t. general terminologies in DLs below $\mathcal{ALC}$. In [9] the problem is shown to remain EXPTIME complete for a DL providing only conjunction, value restriction and existential restriction. The same holds for the small DL $\mathcal{AL}$ which allows for conjunction, value and unqualified existential restriction, and primitive negation [7]. Even for the simple DL $\mathcal{FL}_0$, which only allows for conjunction and value restriction, subsumption w.r.t. cyclic TBoxes with descriptive semantics is PSPACE hard [14], implying hardness for general TBoxes.

Recently, however, it was shown for the DL $\mathcal{EL}$ that subsumption and instance problem w.r.t. cyclic terminologies can be decided in polynomial time [3, 2]. In the present paper we show that even w.r.t. general $\mathcal{ELH}$-TBoxes, including GCIs and simple role inclusion axioms, subsumption and instance problem remain tractable. A surprising result given that DL systems usually employed for reasoning over general terminologies implement—highly optimized—EXPTIME algorithms [13, 10]. Similarly, RACER [10], the only practicable reasoner for ABox reasoning w.r.t. general TBoxes, uses an EXPTIME algorithm for the very expressive DL $\mathcal{ALCNH}_{R^+}$.

The paper is organized as follows. Basic definitions related to general $\mathcal{ELH}$ TBoxes are introduced in Section 2. In Sections 3 and 4 we show how to decide subsumption and instance problem, respectively, w.r.t. general $\mathcal{ELH}$-TBoxes in polynomial time. All details and full proofs of our results can be found in our technical report [4].

## 2 General TBoxes in $\mathcal{ELH}$

*Concept descriptions* are inductively defined with the help of a set of concept *constructors*, starting with a set $N_{\text{con}}$ of *concept names* and a set $N_{\text{role}}$ of *role names*. In this paper, we consider the DL $\mathcal{ELH}$ which provides the concept constructors top-concept ($\top$), conjunction ($C \sqcap D$), and existential restrictions ($\exists r.C$). As usual, $\mathcal{ELH}$ concept descriptions are interpreted w.r.t. a model-theoretic semantics, see [4] for details.

An $\mathcal{EL}$-terminology (called $\mathcal{EL}$-TBox) is a finite set $\mathcal{T}$ of axioms of the form $C \sqsubseteq D$ (called *GCI*) or $C \doteq D$ (called *definition* iff $C \in N_{\text{con}}$) or $r \sqsubseteq s$ (called *simple role inclusion axiom* (SRI)), where $C$ and $D$ are concept descriptions defined in $\mathcal{L}$ and $r, s \in N_{\text{role}}$. A concept name $A \in N_{\text{con}}$ is called *defined in* $\mathcal{T}$ iff $\mathcal{T}$ contains one or more axioms of the form $A \sqsubseteq D$ or $A \doteq D$. The *size* of $\mathcal{T}$ is defined as the sum of the sizes of all axioms in $\mathcal{T}$. Denote by $N_{\text{con}}^{\mathcal{T}}$ the set of all concept names occurring in $\mathcal{T}$ and by $N_{\text{role}}^{\mathcal{T}}$ the set of all role names occurring in $\mathcal{T}$. A TBox that may contain GCIs is called *general*. Denote by $\mathcal{ELH}$ the extension of $\mathcal{EL}$ by SRIs in TBoxes.

An interpretation $\mathcal{I}$ is a *model* of $\mathcal{T}$ iff for every GCI $C \sqsubseteq D \in \mathcal{T}$ it holds that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, for every definition $C \doteq D$ it holds that $C^{\mathcal{I}} = D^{\mathcal{I}}$, and for every SRI $r \sqsubseteq s$ it holds that $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$. A concept description $C$ *subsumes* a concept description $D$ w.r.t. $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{T}$. $C$ and $D$ are *equivalent* w.r.t. $\mathcal{T}$ ($C \equiv_{\mathcal{T}} D$) iff they subsume each other w.r.t. $\mathcal{T}$.

An $\mathcal{ELH}$-ABox is a finite set of assertions of the form $A(a)$ (called *concept assertion*)

or $r(a,b)$ (called *role assertion*), where $A \in N_{\text{con}}$, $r \in N_{\text{role}}$, and $a, b$ are *individual names* from a set $N_{\text{ind}}$. $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ together with an ABox $\mathcal{A}$ iff $\mathcal{I}$ is a model of $\mathcal{T}$ and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ such that all assertions in $\mathcal{A}$ are satisfied, i.e., $a^{\mathcal{I}} \in A^{\mathcal{I}}$ for all $A(a) \in \mathcal{A}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for all $r(a,b) \in \mathcal{A}$. An individual name $a$ is an *instance of* $C$ w.r.t. $\mathcal{T}$ ($\mathcal{A} \models_{\mathcal{T}} C(a)$) iff $a^{\mathcal{I}} \in A^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$ together with $\mathcal{A}$. Denote by $N_{\text{ind}}^{\mathcal{A}}$ the set of all individual names occurring in an ABox $\mathcal{A}$.

The above semantics for TBoxes and ABoxes is usually called *descriptive semantics* [15]. In case of an empty TBox, we write $C \sqsubseteq D$ instead of $C \sqsubseteq_{\emptyset} D$ and analogously $C \equiv D$ instead of $C \equiv_{\emptyset} D$.

**Example 1** As an example of what can be expressed by an $\mathcal{ELH}$-TBox, consider the following TBox showing in an extremely simplified fashion a part of a medical terminology.

$$\begin{aligned}
\mathsf{Pericardium} &\sqsubseteq \mathsf{Tissue} \sqcap \exists \mathsf{cont_in}.\mathsf{Heart} \\
\mathsf{Pericarditis} &\sqsubseteq \mathsf{Inflammation} \\
&\quad \sqcap \exists \mathsf{has_loc}.\mathsf{Pericardium} \\
\mathsf{Inflammation} &\sqsubseteq \mathsf{Disease} \sqcap \exists \mathsf{acts_on}.\mathsf{Tissue} \\
\mathsf{Disease} \sqcap \exists \mathsf{has_loc}.\exists \mathsf{comp_of}.\mathsf{Heart} &\sqsubseteq \mathsf{Heartdisease} \\
&\quad \sqcap \exists \mathsf{is_state}.\mathsf{NeedsTreatment} \\
\mathsf{cont_in} &\sqsubseteq \mathsf{comp_of}
\end{aligned}$$

The TBox contains four GCIs and one SRI, stating, e.g., that Pericardium is tissue contained in the heart and that a diesease located in a component of the heart is a heart disease and requires treatment. Without going into detail, one can check that Pericarditis would be classified as a heart disease requiring treatment because, as stated in the TBox, Pericarditis is a disease located in the Pericardium contained in the heart, and everything contained in something is a component of it.[1]

## 3 Subsumption in $\mathcal{ELH}$ w.r.t. general TBoxes

We aim to show that subsumption of $\mathcal{ELH}$ concepts w.r.t. general TBoxes can be decided in polynomial time. A natural question is whether we may not simply utilize an existing decision procedure for a more expressive DL which might exhibit polynomial time complexity when applied to $\mathcal{ELH}$-TBoxes. Using the standard tableaux algorithm deciding consistency of general $\mathcal{ALC}$-TBoxes [1] as an example, one can show that this approach in general does not bear fruit, even for the sublanguage $\mathcal{EL}$, see [4].

Hence, new techniques are required exploiting the simpler structure of general $\mathcal{ELH}$-TBoxes better. The first step in our approach is to transform TBoxes into a normal form which limits the use of complex concept descriptions to the most basic cases.

[1] The example is only supposed to show the features of $\mathcal{ELH}$ and in no way claims to be adequate from a Medical KR point of view.

**Definition 2** *(Normalized $\mathcal{ELH}$-TBox) Let $\mathcal{T}$ be an $\mathcal{ELH}$-TBox over $N_{\text{con}}$ and $N_{\text{role}}$. $\mathcal{T}$ is* normalized *iff (i) $\mathcal{T}$ contains only GCIs and SRIs, and, (ii) all of the GCIs have one of the following forms:*

$$\begin{aligned} A &\sqsubseteq B \\ A_1 \sqcap A_2 &\sqsubseteq B \\ A &\sqsubseteq \exists r.B \\ \exists r.A &\sqsubseteq B, \end{aligned}$$

*where $A, A_1, A_2, B$ represent concept names from $N_{\text{con}}$ or the top concept $\top$.*

Such a normal form can be computed by exhaustively applying the following transformation rules.

**Definition 3** *(Normalization rules) Let $\mathcal{T}$ be an $\mathcal{ELH}$-TBox over $N_{\text{con}}$ and $N_{\text{role}}$. For every $\mathcal{ELH}$-concept description $C, D, E$ over $N_{\text{role}} \cup \{\top\}$ and for every $r \in N_{\text{role}}$, the $\mathcal{ELH}$-normalization rules are defined modulo commutativity of conjunction ($\sqcap$) as follows:*

$$\begin{aligned}
&\textsf{NF1} && C \doteq D &&\longrightarrow\ \{C \sqsubseteq D,\ D \sqsubseteq C\} \\
&\textsf{NF2} && \hat{C} \sqcap D \sqsubseteq E &&\longrightarrow\ \{\hat{C} \sqsubseteq A,\ A \sqcap D \sqsubseteq E\} \\
&\textsf{NF3} && \exists r.\hat{C} \sqsubseteq D &&\longrightarrow\ \{\hat{C} \sqsubseteq A,\ \exists r.A \sqsubseteq D\} \\
&\textsf{NF4} && C \sqsubseteq \exists r.\hat{D} &&\longrightarrow\ \{C \sqsubseteq \exists r.A,\ A \sqsubseteq \hat{D}\} \\
&\textsf{NF5} && C \sqsubseteq D \sqcap E &&\longrightarrow\ \{C \sqsubseteq D,\ C \sqsubseteq E\}
\end{aligned}$$

*where $\hat{C}, \hat{D}$ denote non-atomic concept descriptions and $A$ denotes a new concept name from $N_{\text{con}}$. Applying a rule $G \longrightarrow \mathcal{S}$ to $\mathcal{T}$ changes $\mathcal{T}$ to $(\mathcal{T} \setminus \{G\}) \cup \mathcal{S}$. The normalized TBox $\text{norm}(\mathcal{T})$ is defined by exhaustively applying Rules* **NF1** *to* **NF4** *and, after that, exhaustively applying Rule* **NF5**.

The size of $\mathcal{T}$ is increased only linearly by exhaustive application of Rule **NF1**. Since this rule never becomes applicable as a consequence of Rules **NF2** to **NF5**, we may restrict our attention to Rules **NF2** to **NF5**. A single application of one of the Rules **NF2** to **NF4** increases the size of $\mathcal{T}$ only by a constant, introducing a new concept name and splitting one GCI into two. Exhaustive application therefore produces an ontology of linear size in the size of $\mathcal{T}$.

After exhaustive application of Rules **NF1** to **NF4**, the left-hand side of every GCI is of constant size. Hence, applying Rule **NF5** exhaustively similarly yields an ontology of linear size in $\mathcal{T}$. Conseqently, the following lemma holds.

**Lemma 4** *The normalized TBox $\text{norm}(\mathcal{T})$ can be computed in linear time in the size of $\mathcal{T}$. The resulting ontology is of linear size in the size of $\mathcal{T}$.*

**ISR** If $s \in S_i(r)$ and $s \sqsubseteq t \in \mathcal{T}$ and $t \notin S_{i+1}(r)$
then $S_{i+1}(r) := S_{i+1}(r) \cup \{t\}$

**IS1** If $A_1 \in S_i(\alpha)$ and $A_1 \sqsubseteq B \in \mathcal{T}$ and $B \notin S_{i+1}(\alpha)$
then $S_{i+1}(\alpha) := S_{i+1}(\alpha) \cup \{B\}$

**IS2** If $A_1, A_2 \in S_i(\alpha)$ and $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T}$
and $B \notin S_{i+1}(\alpha)$ then $S_{i+1}(\alpha) := S_{i+1}(\alpha) \cup \{B\}$

**IS3** If $A_1 \in S_i(\alpha)$ and $A_1 \sqsubseteq \exists r.B \in \mathcal{T}$
and $B_1 \in S_i(B)$ and $s \in S_i(r)$ and $\exists s.B_1 \sqsubseteq C \in \mathcal{T}$
and $C \notin S_{i+1}(\alpha)$ then $S_{i+1}(A) := S_{i+1}(\alpha) \cup \{C\}$

Figure 1: Rules for implication sets

Note that applying Rule **NF5** before exhaustive application of the other rules may produce a terminology of quadratic size in the size of $\mathcal{T}$.

Our strategy is, for every concept name $A \in N_{\text{con}}^{\mathcal{T}}$ and $\top$, to compute a set of concept names $S_*(A)$ with the following property: whenever in some point $x$ in a model of $\mathcal{T}$ the concept $A$ holds then every concept in $S_*(A)$ necessarily also holds in $x$. Similarly, for every role $r$ we want to represent by $S_*(r)$ the set of all roles included in $r$. The simple structure of GCIs in normalized TBoxes allows us to define such sets as follows. To simplify notation, let $N_{\text{con}}^{\mathcal{T},\top} := N_{\text{con}}^{\mathcal{T}} \cup \{\top\}$.

**Definition 5** *(Implication set) Let $\mathcal{T}$ denote a normalized $\mathcal{ELH}$-TBox $\mathcal{T}$ over $N_{\text{con}}$ and $N_{\text{role}}$. For every $A \in N_{\text{con}}^{\mathcal{T},\top}$ ($r \in N_{\text{role}}^{\mathcal{T}}$) and every $i \in \mathbb{N}$, the set $S_i(A)$ ($S_i(r)$) is defined inductively, starting by $S_0(A) := \{A, \top\}$ ($S_0(r) := \{r\}$). For every $i \geq 0$, $S_{i+1}(A)$ ($S_{i+1}(r)$) is obtained by extending $S_i(A)$ ($S_i(r)$) by exhaustive application of the extension rules shown in Figure 1, where $\alpha \in N_{\text{con}}^{\mathcal{T},\top}$. The* implication set *$S_*(A)$ of $A$ is defined as the infinite union $S_*(A) := \bigcup_{i \geq 0} S_i(A)$. Analogously, define $S_*(r) := \bigcup_{i \geq 0} S_i(r)$.*

Note that the successor $S_{i+1}(A)$ of some $S_i(A)$ is generally not the result of only a *single* rule application. $S_{i+1}(A)$ is complete only if no more rules are applicable to any $S_i(B)$ or $S_i(r)$. Implication sets induce a reflexive and transitive but not symmetric relation on $N_{\text{con}}^{\mathcal{T},\top}$ and $N_{\text{role}}^{\mathcal{T}}$, since $B \in S_*(A)$ does not imply $A \in S_*(B)$. We have to show that the idea underlying implication sets is indeed correct. Hence, the occurrence of a concept name $B$ in $S_*(A)$ implies that $A \sqsubseteq_{\mathcal{T}} B$ and vice versa.

**Lemma 6** *For every normalized $\mathcal{ELH}$-TBox over $N_{\text{con}}$ and $N_{\text{role}}$, (i) for every $r, s \in N_{\text{role}}^{\mathcal{T}}$, $s \in S_*(r)$ iff $r \sqsubseteq_{\mathcal{T}} s$, and (ii) for every $A, B \in N_{\text{con}}^{\mathcal{T},\top}$ it holds that $B \in S_*(A)$ iff $A \sqsubseteq_{\mathcal{T}} B$.*

We give a proof sketch, the full proof is shown in [4]. For Claim (i), obviously $r \sqsubseteq_{\mathcal{T}} s$ iff $(r, s)$ is in the transitive closure induced by all $s' \sqsubseteq t' \in \mathcal{T}$. Exactly this closure is computed breadth-first by means of Rule **ISR**.

For the direction ($\Rightarrow$) of Claim (ii), assume $x \in A^{\mathcal{I}}$ for some model $\mathcal{I}$ of $\mathcal{T}$ and $B \in S_*(A)$. Proof by induction over the minimal $n$ with $B \in S_n(A)$. For $n = 0$, $B \in$

$\{A, \top\}$, implying $x \in B^{\mathcal{I}}$. For $n > 0$, we distinguish the rule which caused the inclusion of $B$ in the $i$th step. In each case the induction hypothesis for the precondition of Rule **IS1** to **IS3** implies the semantical consequence $x \in B^{\mathcal{I}}$. For instance, if $B$ has been included in $S_n(A)$ as a result of Rule **IS3** then there exist concept names $A_1, A_2, A_3 \in N_{\text{con}}^{\mathcal{T},\top}$ such that, on the one hand, $A_1 \in S_{n-1}(A)$ and $G := A_1 \sqsubseteq \exists r.A_2 \in \mathcal{T}$, and on the other hand, $A_3 \in S_{n-1}(A_2)$ and $H := \exists s.A_3 \sqsubseteq B \in \mathcal{T}$ with $s \in S_{n-1}(r)$. By induction hypothesis, $r \sqsubseteq_{\mathcal{T}} s$, implying by $G$ that $x \in (\exists r.A_2)^{\mathcal{I}}$. Since $A_3 \in S_{n-1}(A_2)$ the induction hypothesis implies $x \in A_1^{\mathcal{I}}$ and $x \in (\exists s.A_3)^I$, yielding by $H$ that $x \in B^{\mathcal{I}}$.

The reverse direction ($\Leftarrow$) is more involved. We show that if $B \notin S_*(A)$ then there is a model $\mathcal{I}$ of $\mathcal{T}$ with a witness $x_A \in A^{\mathcal{I}} \setminus B^{\mathcal{I}}$. We construct a canonical model $\mathcal{I}$ for $A$ starting from a single vertex $x_A \in A^{\mathcal{I}}$, iteratively applying generation rules which extend $\mathcal{I}$ so as to satisfy all GCIs in $\mathcal{T}$. As $\mathcal{T}$ is normalized, one rule for each type of GCI suffices. For instance, a GCI $A \sqsubseteq \exists r.B$ induces for $x \in A^{\mathcal{I}}$ the creation of an $r$-successor labeled $B$. After showing that the (possibly infinite) model thus constructed is in fact a model of $A$, we show by induction over the construction of $\mathcal{I}$ that the following property holds for every vertex $x$. If $A$ is the first concept name to whose interpretation $x$ was added and if also $x \in B^{\mathcal{I}}$ then $B \in S_*(A)$. Note that this holds in general only if $A$ is the 'oldest' concept with $x \in A^{\mathcal{I}}$. The induction step exploits the fact that if a generation rule for $\mathcal{I}$ forces $x$ into the extension of $B$ then one of the Rules **IS1** to **IS3** includes $B$ into some $S_m(A)$. For instance, in the most simple case, if $x \in B^{\mathcal{I}}$ because of a GCI $C \sqsubseteq B$ then at some point previous, $x \in C^{\mathcal{I}}$, implying $C \in S_*(A)$ by induction hypothesis, yielding $B \in S_*(A)$ by Rule **IS1**.

To show decidability in polynomial time it suffices to show that, (i) $\mathcal{T}$ can be normalized in polynomial time (see above), and, (ii) for all $A \in N_{\text{con}}^{\mathcal{T},\top}$ and $r \in N_{\text{role}}^{\mathcal{T}}$, the sets $S_*(A)$ and $S_*(r)$ can be computed in polynomial time in the size of $\mathcal{T}$. Every $S_{i+1}(A)$ and $S_{i+1}(r)$ depends only on sets with index $i$. Hence, once $S_{i+1}(A) = S_i(A)$ and $S_{i+1}(r) = S_i(r)$ holds for all $A$ and $r$ the complete implication sets are obtained. This happens after a polynomial number of steps, since $S_i(A) \subseteq N_{\text{con}}^{\mathcal{T}}$ and $S_i(r) \subseteq N_{\text{role}}^{\mathcal{T}}$. To compute $S_{i+1}(A)$ and $S_{i+1}(r)$ from the $S_i(B)$ and $S_i(s)$ costs only polynomial time in the size of $\mathcal{T}$.

**Theorem 7** *Subsumption in $\mathcal{ELH}$ w.r.t. general TBoxes can be decided in polynomial time.*

# 4 The instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes

We show that the instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes can be decided in polynomial time. To this end, the approach to decide subsumption by means of implication sets for concept names presented in the previous section is extended to ABox individuals. For every individual name $a \in N_{\text{ind}}^{\mathcal{A}}$, we want to compute a set $S_*(a)$ of concept names with the following property: if $A \in S_*(a)$ then in every model $\mathcal{I}$ of $\mathcal{T}$ together with $\mathcal{A}$ the individual $a^{\mathcal{I}}$ is a witness of $A$ (and vice versa). To extend the definition of implication sets in this way we generalize Rules **IS1** to **IS3** to individual names and introduce a new Rule **IS4** specifically for individual names.

**IS4** If $r(a,b) \in \mathcal{A}$ and $B \in S_i(b)$ and $s \in S_i(r)$
and $\exists s.B \sqsubseteq C \in \mathcal{T}$ and $C \notin S_{i+1}(a)$
then $S_{i+1}(a) := S_{i+1}(a) \cup \{C\}$

Figure 2: Additional rule for implication sets (instance problem)

**Definition 8** *(Implication set) Let $\mathcal{T}$ denote a normalized $\mathcal{ELH}$-TBox $\mathcal{T}$ over $N_{\text{con}}$ and $N_{\text{role}}$ and $\mathcal{A}$ an ABox over $N_{\text{ind}}$, $N_{\text{con}}^{\mathcal{T}}$ and $N_{\text{role}}^{\mathcal{T}}$. For every $r \in N_{\text{role}}^{\mathcal{T}}$, $A \in N_{\text{con}}^{\mathcal{T},\top}$, and $a \in N_{\text{ind}}^{\mathcal{A}}$ and for every $i \in \mathbb{N}$, the sets $S_i(r)$, $S_i(A)$, and $S_i(a)$ are defined inductively, starting by*

$$S_0(r) := \{r\}$$
$$S_0(A) := \{A, \top\}$$
$$S_0(a) := \{A \mid A(a) \in \mathcal{A}\} \cup \{\top\}.$$

*For every $i \geq 0$, $S_{i+1}(r)$, $S_{i+1}(A)$, and $S_{i+1}(a)$ are obtained by extending $S_i(r)$, $S_i(A)$, and $S_i(a)$, respectively, by exhaustive application of Rules* **ISR** *to* **IS4** *shown in Figures 1 and 2, where $\alpha \in N_{\text{con}}^{\mathcal{T},\top} \cup N_{\text{ind}}^{\mathcal{A}}$. The* implication set *$S_*(r)$ of $r$ is defined as the infinite union $S_*(r) := \bigcup_{i\geq 0} S_i(r)$. Analogously, define $S_*(A) := \bigcup_{i\geq 0} S_i(A)$ and $S_*(a) := \bigcup_{i\geq 0} S_i(a)$.*

Since the above definition extends Definition 5 without adding new rules for concept-implication sets $S_*(A)$, Lemma 6 still holds. The following lemma shows that the idea underlying individual-implication sets $S_*(a)$ is also correct in the sense that $A \in S_*(a)$ iff $\mathcal{A} \models_{\mathcal{T}} A(a)$. W.l.o.g. we assume that every individual name $a \in N_{\text{ind}}^{\mathcal{A}}$ has at most one concept assertion $A(a) \in \mathcal{A}$. For every $a$ with $\{A_1(a), A_2(a)\} \subseteq \mathcal{A}$ this can be satisfied by (i) introducing new TBox definitions of the form $A_a \sqsubseteq A_1 \sqcap A_2$ and $A_1 \sqcap A_2 \sqsubseteq A_a$, where $A_a$ is a new concept name, and, (ii) modifying $\mathcal{A}$ to $(\mathcal{A} \setminus \{A_1(a), A_2(a)\}) \cup \{A_a(a)\}$. Iterating this modification yields a normalized TBox $\mathcal{T}'$ of linear size in $\mathcal{T}$ with the required property.

**Lemma 9** *Let $\mathcal{T}$ be a normalized $\mathcal{ELH}$-TBox over $N_{\text{con}}$ and $N_{\text{role}}$ and $\mathcal{A}$ an ABox over $N_{\text{ind}}$, $N_{\text{con}}^{\mathcal{T}}$ and $N_{\text{role}}^{\mathcal{T}}$. For every $A_0 \in N_{\text{con}}^{\mathcal{T}}$ and every $a_0 \in N_{\text{ind}}^{\mathcal{A}}$, $A_0 \in S_*(a_0)$ iff $\mathcal{A} \models_{\mathcal{T}} A_0(a_0)$.*

Similar to Lemma 6, proof direction ($\Rightarrow$) is shown by induction over the least $n$ for which $A_0 \in S_n(a_0)$. For the more interesting reverse direction ($\Leftarrow$), we assume $A_0 \notin S_*(a_0)$ and construct a canonical model $\mathcal{I}$ of $\mathcal{T}$ together with $\mathcal{A}$ where $a^{\mathcal{I}} \notin A^{\mathcal{I}}$. See [4] for the full proof.

The proof of decidability in polynomial time is analogous to the case of subsumption: regarding computational complexity, the individual-implication sets $S_*(a)$ have the same properties as concept-implication sets. The new Rule **IS4** also does not increase the complexity of computing the sets $S_*(a)$ significantly.

**Theorem 10** *The instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes can be decided in polynomial time.*

# 5 Conclusion

We have seen how subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes can be decided in polynomial time. Moreover, the implication sets computed for one TBox $\mathcal{T}$ can be used to decide *all* subsumptions between defined (or primitive) concepts in $\mathcal{T}$. Hence, classifying $\mathcal{T}$ requires only a single computation of the implication sets for $\mathcal{T}$. The same holds for the instance problem, where a single computation of the relevant implication sets suffices to classify $\mathcal{T}$ and decide *all* instance problems w.r.t. defined (or primitive) concepts occurring in $\mathcal{T}$.

Since subsumption and instance problem remain tractable under the transition from cyclic to general $\mathcal{EL}$-TBoxes, the second natural question is how far the DL can be extended further preserving tractability. Obviously, adding value restrictions makes subsumption NP hard even for the empty TBox [8]. Moreover, it can be shown that adding one of the constructors number restriction, disjunction, or allsome [6] makes subsumption co-NP hard even without GCIs.

It is open, however, whether subsumption and instance problem w.r.t. general TBoxes remain tractable when extending $\mathcal{ELH}$ by inverse roles. Extending our subsumption algorithm by more expressive role constructors might lead the way to a more efficient reasoning algorithm for the representation language underlying the GALEN [17] terminology, where inverse roles and complex role inclusion axioms can be expressed. While the polynomial upper bound would undoubtedly be exceeded, still a complexity better than EXPTIME might be feasible.

### Acknowledgements

My thanks to Carsten Lutz for a multitude of useful remarks and ideas that have greatly influenced this work.

## References


[1] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[2] Franz Baader. The instance problem and the most specific concept in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI 2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Hamburg, Germany, 2003. Springer-Verlag.

[3] Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 325–330. Morgan Kaufmann, 2003.

[4] S. Brandt. Subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general tboxes. LTCS-Report LTCS-04-04, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2004. See http://lat.inf.tu-dresden.de/research/reports.html.

[5] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[6] Robert Dionne, Eric Mays, and Frank J. Oles. The equivalence of model-theoretic and structural subsumption in description logics. In Ruzena Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 710–716, San Mateo, California, 1993. Morgan Kaufmann.

[7] F.M. Donini. Complexity of reasoning. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 96–136. Cambridge University Press, 2003.

[8] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Bernhard Hollunder, Werner Nutt, and Alberto Spaccamela. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53(2–3):309–327, 1992.

[9] Robert Givan, David A. McAllester, Carl Witty, and Dexter Kozen. Tarskian set constraints. *Information and Computation*, 174(2):105–131, 2002.

[10] Volker Haarslev and Ralf Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701–712, 2001.

[11] Ian Horrocks, Alan L. Rector, and Carole A. Goble. A description logic based schema for the classification of medical data. In *Knowledge Representation Meets Databases*, 1996.

[12] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, September 1999.

[13] Ian R. Horrocks. Using an expressive description logic: FaCT or fiction? In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 636–645. Morgan Kaufmann, San Francisco, California, 1998.

[14] Yevgeny Kazakov and Hans De Nivelle. Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is pspace-complete. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, CEUR-WS, 2003.

[15] B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.

[16] A. Rector. Medical informatics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 406–426. Cambridge University Press, 2003.

[17] A. Rector, S. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.

[18] A. Rector, W. Nowlan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proceedings of the 17th annual Symposium on Computer Applications in Medical Care, Washington, USA, SCAMC*, pages 414–418, 1993.

# The Instance Store: DL Reasoning with Large Numbers of Individuals

Ian Horrocks, Lei Li, Daniele Turi and Sean Bechhofer
University of Manchester, UK
<lastname>@cs.man.ac.uk

**Abstract**

We present an application – the Instance Store – aimed at solving some of the scalability problems that arise when reasoning with the large numbers of individuals envisaged in the semantic web. The approach uses well-known techniques for reducing description logic reasoning with individuals to reasoning with concepts. Crucial to the implementation is the combination of a description logic terminological reasoner with a traditional relational database. The resulting form of inference, although specialised, is sound and complete and sufficient for several interesting applications. Most importantly, the application scales to sizes (over 100,000s individuals) where all other existing applications fail. This claim is substantiated by a detailed empirical evaluation of the Instance Store in contrast with existing alternative approaches.

## Introduction

The Semantic Web [6] aims at making Web resources more accessible to automated processes by adding "semantic annotations"—metadata that describes their content. It is envisaged that the semantics in semantic annotations will be given by ontologies, which will provide a source of precisely defined terms (vocabularies) that are amenable to automated reasoning.

A standard for expressing ontologies in the Semantic Web has already emerged: the ontology language *OWL* [9], which recently became a W3C recommendation. One of the main features of OWL is that there is a direct correspondence between (two of the three "species" of) OWL and *Description Logics* (*DL*s) [19]. This means that DL reasoners can be used to reason about OWL ontologies and about annotations that are instances of concept descriptions formed using terms from an ontology.

Unfortunately, while existing techniques for *TBox* reasoning (i.e., reasoning about the concepts in an ontology) seem able to cope with real world ontologies [18, 14], it is not clear if existing techniques for *ABox* reasoning (i.e., reasoning about the individuals in an ontology) will be able to cope with realistic sets of instance data. This difficulty arises not so much from the computational complexity of ABox reasoning, but from the fact that the number of individuals (e.g., annotations) might be extremely large.

In this paper we describe the *instance Store* (*iS*), an approach to a restricted form of ABox reasoning that combines a DL reasoner with a database. The result is a system that can deal with very large ABoxes, and is able to provide sound and complete answers to instance retrieval queries (i.e., computing all the instances of a given query concept) over such ABoxes.

While *iS* can be highly effective, it does have limitations when compared to a fully fledged DL ABox. In particular, *iS* can only deal with a *role-free* ABox, i.e., an ABox that does not contain any axioms asserting role relationships between pairs of individuals. Although this may seem a rather severe restriction, the functionality provided by *iS* is precisely what is required by many applications, and in particular by applications where ontology based terms are used to describe/annotate and retrieve large numbers of objects. Examples include the use of ontology based vocabulary to describe documents in "publish and subscribe" applications [10], to annotate data in bioinformatics applications [12] and to annotate web resources such as web pages [11] or web service descriptions [20] in Semantic Web applications. Indeed, we have successfully applied *iS* to perform web service discovery [8], to search over the gene ontology [12] and its associated instances (see below), and in an application to guide gene annotation [4].

Using a database in order to support ABox reasoning is certainly not new (see below), but to the best of our knowledge *iS* is the first such system that is general purpose (i.e., can deal with any TBox and role-free ABox without customising the database schema), provides sound and complete reasoning, and places no a-priori restriction on the size of the ABox.

In order to evaluate the design of *iS*, and in particular its ability to provide scalable performance for instance retrieval queries, we have performed a number of experiments using *iS* to search over a large (50,000 concept) gene ontology and its associated very large number (up to 650,000) of individuals – instances of concept descriptions formed using terms from the ontology. In the absence of other specialised ABox reasoners we have compared the performance of *iS* with that of RACER [15] (the only publicly available DL system that supports full ABox reasoning for an expressive DL) and of FaCT [18] (using TBox reasoning to simulate reasoning with a role-free ABox).

**Related Work** As already mentioned, the idea of supporting DL style reasoning using databases is not new. One example is [7], which can handle DL inference problems by converting them into a collection of SQL queries. This approach is not limited to role-free ABoxes, but the DL language supported is much less expressive, and the database schema must be customised according to the given TBox. Another example is the Parka system [2]. Parka is not limited to role-free ABoxes and can deal with very large ABoxes. However, Parka also supports a much less expressive language, and is not based on standard DL semantics, so it is not really comparable to *iS*. Finally, [21] describes a "semantic indexing" technique that is very similar to the approach used in *iS* except that files and hash tables are used instead of database tables, and optimisations such as the use of equivalence sets are not considered.

# 1 Instance Store

Description Logics are a family of knowledge representation formalisms evolved from early *frame systems* and *semantic networks*. We assume the reader to be familiar with DLs—see [3] for a detailed discussion of DLs.

An ABox $\mathcal{A}$ is role-free if it contains only axioms of the form $x : C$. We can assume, without loss of generality, that there is exactly one such axiom for each individual as $x : C \sqcup \neg C$ holds in all interpretations, and two axioms $x : C$ and $x : D$ are equivalent to a single

axiom $x : (C \sqcap D)$. It is well known that, for a role-free ABox, instantiation can be reduced to TBox subsumption [16, 22]; i.e., if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, and $\mathcal{A}$ is role-free, then $\mathcal{K} \models x : D$ iff $x : C \in \mathcal{A}$ and $\mathcal{T} \models C \sqsubseteq D$. Similarly, if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and $\mathcal{A}$ is a role-free ABox, then the instances of a concept $D$ could be retrieved simply by testing for each individual $x$ in $\mathcal{A}$ if $\mathcal{K} \models x : D$. However, this would clearly be very inefficient if $\mathcal{A}$ contained a large number of individuals.

An alternative approach is to add a new axiom $C_x \sqsubseteq D$ to $\mathcal{T}$ for each axiom $x : D$ in $\mathcal{A}$, where $C_x$ is a new atomic concept; we will call such concepts *pseudo-individuals*. Classifying the resulting TBox is equivalent to performing a complete realisation of the ABox: the most specific atomic concepts that an individual $x$ is an instance of are the most specific atomic concepts that subsume $C_x$ and that are not themselves pseudo-individuals. Moreover, the instances of a concept $D$ can be retrieved by computing the set of pseudo-individuals that are subsumed by $D$. The problem with this latter approach is that the number of pseudo-individuals added to the TBox is equal to the number of individuals in the ABox, and if this number is very large, then TBox reasoning may become inefficient or even break down completely (e.g., due to resource limits).

The basic idea behind *iS* is to overcome this problem by using a DL reasoner to classify the TBox and a database to store the ABox, with the database also being used to store a complete realisation of the ABox, i.e., for each individual $x$, the concepts that $x$ realises (the most specific atomic concepts that $x$ instantiates). The realisation of each individual is computed using the DL (TBox) reasoner when an axiom of the form $x : C$ is added to the *iS* ABox.

A retrieval query $Q$ to *iS* (i.e., computing the set of individuals that instantiate a concept $Q$) can be answered using a combination of database queries and TBox reasoning. Given an *iS* containing a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a query concept $Q$, retrieval involves the computation of sets of concepts and individuals which we denote as follows:

- $Q\downarrow_{\mathcal{T}}$ denotes the set of atomic concepts in $\mathcal{T}$ subsumed by $Q$; these are the equivalents and descendants of $Q$ in $\mathcal{T}$.
- $\lceil Q \rceil_{\mathcal{T}}$ denotes the set of most specific atomic concepts in $\mathcal{T}$ subsuming $Q$; if $Q$ is itself an atomic concept in $\mathcal{T}$ then clearly $\lceil Q \rceil_{\mathcal{T}} = \{Q\}$.
- $I_1$ denotes the set of individuals in $\mathcal{A}$ that realise *some* concept in $Q\downarrow_{\mathcal{T}}$;
- $I_2$ denotes the set of individuals in $\mathcal{A}$ that realise *every* concept in $\lceil Q \rceil_{\mathcal{T}}$.

The *iS* algorithm to retrieve the instances of $Q$ can be then described as follows:

1. use the DL reasoner to compute $Q\downarrow_{\mathcal{T}}$;
2. use the database to find the set of individuals $I_1$;
3. use the reasoner to check whether $Q$ is equivalent to any atomic concept in $\mathcal{T}$; if that is the case then simply return $I_1$ and *terminate*;
4. otherwise, use the reasoner to compute $\lceil Q \rceil_{\mathcal{T}}$;
5. use the database to compute $I_2$;

6. use the reasoner and the database to compute $I_3$, the set of individuals $x \in I_2$ such that $x : C$ is an axiom in $\mathcal{A}$ and $C$ is subsumed by $Q$;

7. return $I_1 \cup I_3$ and *terminate*.

**Proposition.** The above procedure is sound and complete for retrieval, i.e., given a concept $Q$, it returns all and only individuals in $\mathcal{A}$ that are instances of $Q$.

The above is easily proved using the fact that we assume, without loss of generality, that for each individual there is only one axiom associated to it.

### An Optimised Instance Store

In practice, several refinements to the above procedure are used to improve the performance of *iS*. In the first place, as it is potentially costly, we should try to minimise the DL reasoning required in order to compute realisations (when instance axioms are added to the ABox) and to check if individuals in $I_1$ are instances of the query concept (when answering a query).

One way to (possibly) reduce the need for DL reasoning is to avoid repeating computations for "equivalent" individuals, e.g., individuals $x_1, x_2$ where $x_1 : C_1$ and $x_2 : C_2$ are ABox axioms, and $C_1$ is equivalent to $C_2$. Since checking for semantic equivalence between two concepts would require DL reasoning (which we are trying to avoid), the optimised *iS* only checks for syntactic equality using a database lookup. (The chances of detecting equivalence via syntactic checks could be increased by transforming concepts into a syntactic normal form, as is done by optimised DL reasoners [17], but this additional refinement has not yet been implemented in *iS*.) Individuals are grouped into equivalence sets, where each individual in the set is asserted to be an instance of a syntactically identical concept, and only one representative of the set is added to the *iS* ABox as an instance of the relevant concept. When answering queries, each individual in the answer is replaced by its equivalence set. Similarly, we can avoid repeated computations of sub and super-concepts for the same concept (e.g., when repeating a query) by caching the results of such computations in the database.

Finally, the number and complexity of database queries also has a significant impact on the performance of *iS*. In particular, the computation of $I_1$ can be costly as $Q \downarrow_{\mathcal{T}}$ may be very large. One way to reduce this complexity is to store not only the most specific concepts instantiated by each individual, but to store *every* concept instantiated by each individual. As most concept hierarchies are relatively shallow, this does not increase the storage requirement too much, and it greatly simplifies the computation of $I_1$: it is only necessary to compute the (normally) much smaller set of most general concepts subsumed by $Q$ and to query the database for individuals that instantiate some member of such set. On the other hand, the computation of $I_2$ is slightly more complicated, because $I_1$ must be subtracted from the set of individuals that instantiate every concept in $\lceil Q \rceil_{\mathcal{T}}$. Empirically, however, the savings when computing $I_1$ seems to far outweigh the extra cost of computing $I_2$.

## 2 Implementation

We have implemented *iS* using a component based architecture that is able to exploit existing DL reasoners and databases. The core component is a Java application [1] talking to a DL

reasoner via the DIG interface [5] and to a relational database via JDBC. We have tested it with FaCT [18] and RACER reasoners and MySQL, Hypersonic, and Oracle databases.

```
initialise(Reasoner reasoner, Database db, TBox t)
addAssertion(Individual i, Concept C)
retract(Individual i)
retrieve(Concept Q): Set⟨Individual⟩
```

Figure 1: Basic functionality of *iS*

The basic functionality of *iS* is illustrated by Figure 1. The four basic operations are `initialise`, which loads a TBox into the DL reasoner, classifies the TBox and establishes a connection to the database; `addAssertion`, which adds an axiom $i : D$ to *iS*; `retract`, which removes any axiom of the form $i : C$ (for some concept $C$) from *iS*; and `retrieve`, which returns the set of individuals that instantiate a query concept $Q$. Since an *iS* ABox can only contain one axiom for each individual, asserting $i : D$ when $i : C$ is already in the ABox is equivalent to first removing $i$ and then asserting $i : (C \sqcap D)$.

In the current implementation, we make the simplifying assumption that the TBox itself does not change. Extending the implementation to deal with monotonic extensions of the TBox would be relatively straightforward, but deleting information from the TBox might require (in the worst case) all realisations to be recomputed.

**Database.** For the basic *iS*, the database schema is straightforward: a table with all the assertions stored as pairs individual/concept (with individual as primary key), and a table of pairs individual/*atomic* concept. The latter table holds the asserted individuals together with the most specific atomic concepts instantiated by them.

For the optimised *iS*, the database schema is illustrated in Figure 2. There is a main

```
Concepts(id, concept)
Assertions(individual, conceptId)
Types(conceptId, atomicConcept)
Equivalents(conceptId, atomicConcept)
Parents(conceptId, atomicConcept)
Children(conceptId, atomicConcept)
```

Figure 2: Database Schema for the Optimised *iS*

`Concepts` table assigning a unique `id` to every asserted or retrieved concept; the `conceptId` in the other tables is a foreign key referencing `id`. Apart from the evident `Assertions` table, the remaining tables hold TBox information inferred using the reasoner: the `Types` table holds all ancestors and equivalents of the asserted/retrieved concepts, while the position of the concepts in the taxonomy is recorded by either storing their equivalents if they exist or both their children and parents in the corresponding tables.

# 3 Empirical Evaluation

To illustrate the scalability and performance of *iS* we describe the tests we have performed using the gene ontology and its associated instance data. We also illustrate how this compares

with existing non-specialised ABox reasoning techniques by describing the same tests performed using RACER and FaCT (the latter using the pseudo-individual approach discussed in Section 1).

The gene ontology (*GO*) itself, an ontology describing terms used in gene products and developed by the Gene Ontology Consortium [23], is little more than three taxonomies of gene terms, with a single role being used to add "part-of" relationships. However, the ontology is large (47,012 atomic concepts) and the instance data, obtained by mining the GO database [13] of gene products, consists of 653,762 individual axioms involving 48,581 distinct complex DL expressions using three more roles.

The retrieval performance tests use two sets of queries. The first set (Q1-Q5) was formulated with the help of domain experts and consists of five realistic queries that might be posed by a biologist. The second set (Q6-Q11) consists of six artificial queries designed to test the effect on query answering performance of factors such as the number of individuals in the answer, whether the query concept is equivalent to an atomic concept (if so, then the answer can be returned without computing $I_3$), and the number of candidate individuals in $I_2$ for which DL reasoning is required in order to determine if they form part of the answer. The characteristics of the various queries with respect to these factors is shown in Table 1.

Table 1: Query characteristics

| Query | Equivalent to Atomic Concept | No. of Instances in Answer | No. of "candidates" in $I_2$ |
|---|---|---|---|
| Q1 | Yes | 2,641 | n/a |
| Q2 | No | 0 | 284 |
| Q3 | No | 3 | 284 |
| Q4 | Yes | 7,728 | n/a |
| Q5 | Yes | 25 | n/a |
| Q6 | No | 13,449 | 551 |
| Q7 | No | 11,820 | 116 |
| Q8 | No | 12 | 603 |
| Q9 | No | 19 | 19 |
| Q10 | Yes | 4,543 | n/a |
| Q11 | Yes | 1 | n/a |

### 3.1 Loading and Querying Tests

In these tests, we compared the performance of *iS* with that of RACER using the GO TBox and different sized subsets of the GO ABox. The *iS* was first initialised with the GO TBox, then for each ABox, we measured the time (in CPU seconds) taken to load the ABox into it. A comparison with RACER is shown in Table 2.

The time taken by the *iS* to load the ABoxes increases more slowly than their size: for ABox size 200, *iS* takes about 1s to add each individual axiom; by the time the ABox size has reached 400,000 this has fallen to approximately 0.25s per axiom. In view of the equivalent individuals optimisation employed by *iS*, however, it may be more relevant to consider the

Table 2: *iS* and RACER load and realise times (CPU seconds)

| Number of Individuals | Distinct Descriptions | Load & Realise (s) | |
|---|---|---|---|
| | | *iS* | RACER |
| 200 | 155 | 189 | 180 |
| 500 | 330 | 405 | 3,420 |
| 1,000 | 591 | 804 | 22,320 |
| 2,000 | 1,017 | 1,395 | fault |
| 5,000 | 2,024 | 2,906 | fault |
| 10,000 | 3,299 | 5,988 | fault |
| 20,000 | 5,364 | 11,057 | fault |
| 50,000 | 9,760 | 21,579 | fault |
| 100,000 | 15,147 | 33,456 | fault |
| 200,000 | 23,387 | 56,613 | fault |
| 400,000 | 35,800 | 96,503 | fault |
| 653,762 | 48,581 | 140,623 | fault |

time taken per distinct description: this increases from about 1s per description for the size 200 ABox (which contains 155 distinct descriptions) to approximately 3s per description for the size 653,762 ABox (which contains 48,581 distinct descriptions).

The time taken by RACER to realise the smallest ABox is roughly the same as that taken by *iS*. As the ABox size grows, however, the time taken by RACER increases rapidly, and at ABox size 1,000 it is already taking approximately 22s per axiom. For larger ABoxes, RACER broke down due to a resource allocation error in the underlying Lisp system.

Next, we measured retrieval times. For RACER, we carried out the same tests in two different ways. In both cases we first initialised RACER with the GO TBox, then loaded the ABox. In the first test, we used the *realize-abox* function to force RACER to compute a complete realisation of the ABox before answering any queries; if the realisation was successfully completed, we then timed how long it took to answer each of the queries. In the second test, we simply timed how long it took RACER to answer each of the queries without first forcing it to realise the ABox.

The results for *iS* when answering each of the five realistic queries and six artificial queries are plotted against the size of the ABox in Figure 3; note the logarithmic scales on both axes. The figure shows that the time taken to answer queries like Q6 and Q8 becomes quite large. This is due to the fact that $I_2$ is large, thus demanding repeated invocations of the expensive check (roughly 0.2s per individual) in step 6 of *iS* retriecal algorithm. The number of "distinct" individuals in the answer also has a significant impact on performance: when there are many such individuals, the database query required in order to compute the complete answer set can be quite time consuming.

We tested also RACER with the above queries, both in the case where the ABox has been realised and where it has not. Once the ABox has been realised, queries are answered almost *instantly*, but results are only available for the relatively small ABoxes that RACER was able to realise (up to 1,000 individuals). When the ABox was not realised, answers

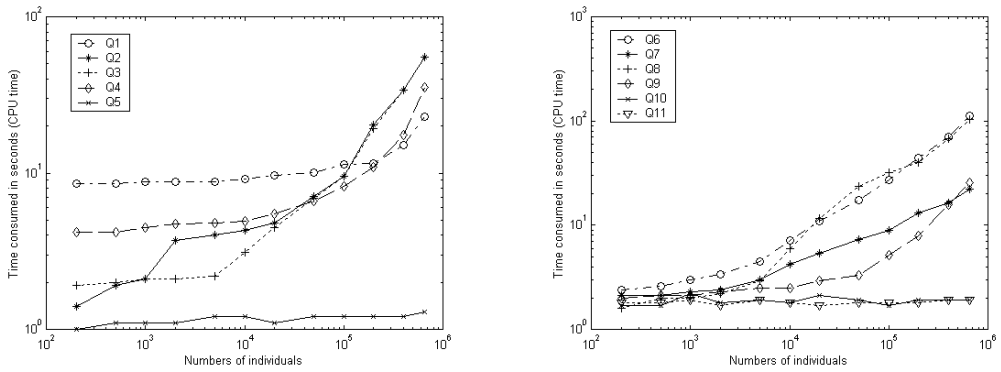



Figure 3: *iS* realistic (left) and artificial (right) query times -v- ABox size

were again returned almost *instantly* for smaller ABoxes, but when the ABox size exceeded 1,000 individuals the answer times increased dramatically, and for ABoxes larger than 10,000 individuals (larger than 5,000 in the case of Q9) RACER again broke down due to a resource allocation error in the underlying Lisp system.

It should be mentioned that the results for *iS* include significant communication overheads (both with the database and DL reasoner), which was not the case for RACER since queries were posed directly via its command line interface.

### 3.2 Pseudo-individual Tests

As discussed in §1, one way to deal with role-free ABoxes is to treat individuals as atomic concepts in the TBox (pseudo-individuals). To test the feasibility of this approach, we again used the GO TBox and ABox, and the set of queries described above. To make the comparison fair, only distinct instantiated concept are used. The FaCT system was used in these tests as RACER broke down when trying to classify the GO TBox augmented with the pseudo-individuals, again due to a resource allocation error in the underlying Lisp system.

In order to investigate how the pseudo-individual approach would scale with increasing ABox (and hence TBox) size, we tried computing the concepts subsumed by each query with the GO TBox alone (which contains 47,012 concept names) and with the TBox augmented with the pseudo-individuals derived from the GO ABox (a total of 95,593 concept names). The results of these tests are given in Table 3. It is important to note that they do not include the time required to expand answers to include sets of equivalent individuals—as discussed above, this can be quite time consuming for some queries (e.g., 19.5s in the case of Q9 with the largest ABox).

As one can see, the time taken to compute the answers to the queries is heavily dependent on the size of the answers, and in the case of Q4 with the pseudo-individual augmented TBox, the time was over 600s. This is in contrast to *iS*, where the size of answer had comparatively little effect on the time taken to answer queries. For queries with relatively small answers, however, the pseudo-individual approach was highly effective, even for queries that were time consuming to answer using *iS*.

Table 3: Pseudo-individual query time (CPU seconds) and answer size

| Query | GO TBox | | GO TBox + ABox | |
|---|---|---|---|---|
| | Time | Answer Size | Time | Answer Size |
| Q1 | 8.1 | 220 | 233.3 | 2,861 |
| Q2 | 1.3 | 1 | 1.2 | 1 |
| Q3 | 0.2 | 1 | 1.4 | 4 |
| Q4 | 26.0 | 881 | 631.8 | 8,609 |
| Q5 | 0.5 | 2 | 5.2 | 27 |
| Q6 | 4.3 | 86 | 176.6 | 2,450 |
| Q7 | 1.4 | 1 | 10.0 | 147 |
| Q8 | 1.3 | 1 | 1.5 | 7 |
| Q9 | 1.4 | 1 | 3.5 | 22 |
| Q10 | 4.2 | 109 | 114.4 | 1,407 |
| Q11 | 0.5 | 1 | 2.0 | 2 |

# 4 Discussion and Future Work

Our experiments show that *iS* provides stable and effective reasoning for role-free ABoxes, even those containing very large numbers of individuals. In contrast, full ABox reasoning using the RACER system exhibited accelerating performance degradation with increasing ABox size, and at least the current RACER release (1.7.7) was not able to deal with the larger ABoxes used in our evaluation. The pseudo-individual approach to role-free ABox reasoning was more promising, and may be worth further investigation.

The acceptability of the performance of *iS* would obviously depend on the nature of the application and the characteristics of the KB and of typical queries. It is likely that the performance of *iS* can be substantially improved simply by dealing with constant factors such as communication overheads.

Future work includes the investigation of additional optimisations and enhancements, such as providing a more sophisticated query interface. We are also investigating ways to extend *iS* to ABoxes that are not completely role-free. This may be possible in restricted cases by applying some form of *precompletion* [16] to the ABox.

### Acknowledgements.

Thanks to Phil Lord for help with the implementation and to Chris Wroe for help with the GO ontology and the formulation of realistic queries.

# References

[1] Instance Store website. `http://instancestore.man.ac.uk`.

[2] W. A. Andersen, K. Stoffel, and J. A. Hendler. Parka: Support for extremely large knowledge bases. In G. Ellis et al, editor, *Proc. First Int'l KRUSE Symposium*, pages 122–133, 1995.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. CUP, 2003.

[4] M. Bada, D. Turi, R. McEntire, and R. Stevens. Using Reasoning to Guide Annotation with Gene Ontology Terms in GOAT. *SIGMOD Record (special issue on data engineering for the life sciences)*, June 2004.

[5] Sean Bechhofer. The DIG description logic interface: DIG/1.1. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, 2003.

[6] Tim Berners-Lee. *Weaving the Web*. Harpur, San Francisco, 1999.

[7] Alexander Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 217–226, 1993.

[8] Olga Caprotti, Mike Dewar, and Daniele Turi. Mathematical service matching using Description Logic and OWL. Technical Report IST-2001-34145, MONET Consortium, March 2004.

[9] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. OWL web ontology language 1.0 reference, July 2002.

[10] M. Uschold et al. A semantic infosphere. In D. Fensel et al, editor, *Proc. ISWC 2003*, number 2870 in LNCS, pages 882–896. Springer, 2003.

[11] Stephen Dill et al. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proc. WWW 2003*, 2003.

[12] GO project. European Bioinformatics Institute. `http://www.ebi.ac.uk/go`.

[13] Gene Ontology Database, 2003. `http://www.godatabase.org/dev/database/`.

[14] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. IJCAI 2001*, 2001.

[15] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. IJCAR 2001*, volume 2083 of *LNAI*, pages 701–705. Springer, 2001.

[16] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Ann. of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996.

[17] I. Horrocks. Implementation and optimisation techniques. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. CUP, 2003.

[18] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[19] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2nd International Semantic Web Conference (ISWC)*, 2003.

[20] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. WWW 2003*, pages 331–339. ACM, 2003.

[21] A. Schmiedel. Semantic indexing based on description logics. In F. Baader, M. Buchheit, M.A. Jeusfeld, and W. Nutt, editors, *Reasoning about structured objects: knowledge representation meets databases. Proceedings of the KI'94 Workshop KRDB'94*, September 1994.

[22] Sergio Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, University of Manchester, Department of Computer Science, April 2001.

[23] The Gene Ontology Consortium. Gene ontolgy: Tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.

# Efficient Reasoning with Range and Domain Constraints

Dmitry Tsarkov and Ian Horrocks

Department of Computer Science
The University of Manchester
Manchester, UK
{tsarkov|horrocks}@cs.man.ac.uk

**Abstract**

We show how a tableaux algorithm for $\mathcal{SHIQ}$ can be extended to support role boxes that include range and domain axioms, prove that the extended algorithm is still a decision procedure for the satisfiability and subsumption of $\mathcal{SHIQ}$ concepts w.r.t. such a role box, and show how support for range and domian axioms can be exploited in order to add a new form of absorption optimisation called role absorption. We illustrate the effectiveness of the optimised algorithm by analysing the perfomance of our FaCT++ implementation when classifying terminologies derived from realistic ontologies.

## 1 Introduction

Many modern ontology languages (e.g., OIL [3], DAML+OIL [8] and OWL [2]) are based on expressive description logics, and in particular on the $\mathcal{SHIQ}$ family of description logics [9]. These ontology languages typically support domain and range constraints on roles, i.e., axioms asserting that if an individual $x$ is related to an individual $y$ by a role $R$, then $x$ must be an instance of the concept that is the domain of $R$ and $y$ must be an instance of the concept that is the range of $R$.Such axioms are not directly supported by $\mathcal{SHIQ}$, but can trivially be transformed into *general inclusion axioms* (GCIs), i.e., an axiom asserting a subsumption relationship between two arbitrary concept terms. In particular, restricting the domain of a role $R$ to be concept $C$ is equivalent to adding an axiom $\exists R.\top \sqsubseteq C$, and restricting the range of a role $R$ to be concept $D$ is equivalent to adding an axiom $\top \sqsubseteq \forall R.D$.

The problem with this transformation is that such GCIs are not amenable to *absorption*, an optimisation technique that tries to rewrite GCIs so that they can be efficiently dealt with using the *lazy unfolding* optimisation [6]. Absorption is one of the crucial optimisations that enable state of the art DL reasoners such as FaCT [7], Racer [5] and Pellet [12] to deal effectively with large knowledge bases (KBs), and these reasoners perform much less well with KBs containing significant numbers of unabsorbable GCIs. Unfortunately, many ontologies contain large numbers of different roles, each with a range and domain constraint, and the resulting KBs therefore contain many unabsorbable GCIs.

It has already been shown that, in order for the Racer system to be able to classify large KBs containing many range and domain constraints, it is necessary to give a special treatment to the GCIs introduced by range and domain axioms [4]. The approach used by Racer is to extend the lazy unfolding optimisation so that concepts equivalent to those that would be

introduced by the GCIs are introduced only as necessary. In the approach presented here, we extend the tableaux satisfiability testing algorithm so that range and domain axioms are directly supported. The advantage with this approach is that we are able to extend the formal correctness proof to demonstrate that the extended algorithm is still a decision procedure for $\mathcal{SHIQ}$ satisfiability (i.e., it returns *satisfiable* iff the input concept is satisfiable).

As well as allowing range and domain to be dealt with very efficiently, this algorithm also allows us to implement an extended version of the absorption optimisation, called *role absorption*, that transforms GCIs into domain constraints. Role absorption can provide alternative and perhaps more effective ways to absorb certain forms of GCI, and can also be applied to some otherwise unabsorbable forms of GCI. This can lead to dramatic performance improvements for KBs that contain significant numbers of such GCIs. We demonstrate this (as well as the performance improvements resulting from support for range and domain axioms) with an empirical analysis of the performance of the extended algorithm when classifying several KBs derived from realistic ontologies.

## 2 Preliminaries

We first introduce the syntax and semantics of the $\mathcal{SHIQ}$ logic, including the semantics of role boxes extended with range and domain axioms. Most details of the logic and the tableaux algorithm are little changed from those presented in [9]. We will, therefore, focus mainly on the parts that have been added in order to deal with range and domain axioms, and refer the reader to [9] for complete information on the remainder.

The absolutely most part of formal definitions here is taken from [9]. We have introduced new constructions into the existing definitions, so all algorithms were slightly changed.

**Definition 1** *Let* $\mathbf{C}$ *and* $\mathbf{R}$ *be disjoint sets of* concept names *and* role names *respectively. The set of* $\mathcal{SHIQ}$-roles *is* $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. *To avoid considering roles such as* $R^{--}$*, we define a function* $\mathsf{Inv}$ *on roles such that* $\mathsf{Inv}(R) = R^-$ *if* $R$ *is a role name, and* $\mathsf{Inv}(R) = S$ *if* $R = S^-$. *For* $R$ *and* $S$ $\mathcal{SHIQ}$*-roles and* $C$ *a* $\mathcal{SHIQ}$*-concept, a* role axiom *is either a role inclusion of the form* $R \sqsubseteq S$*, a transitivity axiom of the form* $\mathsf{Trans}(R)$*, or a* constraint axiom *of the form* $\mathsf{Domain}(R, C)$ *or* $\mathsf{Range}(R, C)$. *A* role box $\mathcal{R}$ *is a finite set of role axioms.*

*A role* $R$ *is called* simple *if, for* $\sqsubseteq^*$ *the transitive reflexive closure of* $\sqsubseteq$ *on* $\mathcal{R}$ *and for each role* $S$, $S \sqsubseteq^* R$ *implies* $\mathsf{Trans}(S) \notin \mathcal{R}$ *and* $\mathsf{Trans}(\mathsf{Inv}(S)) \notin \mathcal{R}$.

*The set of* concepts *is the smallest set such that every concept name is a concept, and, for* $C$ *and* $D$ *concepts,* $R$ *a role,* $S$ *a simple role and* $n$ *a non-negative integer, then* $C \sqcap D$, $C \sqcup D$, $\neg C$, $\exists R.C$, $\forall R.C$, $\geqslant nS.C$ *and* $\leqslant nS.C$ *are also concepts.*

*The semantics is given by means of an* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consisting of a non-empty set* $\Delta^{\mathcal{I}}$*, called the* domain *of* $\mathcal{I}$*, and a* valuation $\cdot^{\mathcal{I}}$ *which maps every concept to a subset of* $\Delta^{\mathcal{I}}$ *and every role to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *such that, for all concepts* $C$, $D$, *roles* $R$, $S$, *and non-negative integers* $n$, *the properties in Figure 1 are satisfied, where* $\sharp M$ *denotes the cardinality of a set* $M$.

*An interpretation satisfies a role axiom if it satisfies the semantic conditions given in Figure 1. An interpretation satisfies a role box* $\mathcal{R}$ *if it satisfies each role axiom in* $\mathcal{R}$.

*A* terminology *or* TBox $\mathcal{T}$ *is a finite set of* general concept inclusion axioms, $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$, *where* $C_i, D_i$ *are arbitrary* $\mathcal{SHIQ}$*-concepts. An interpretation* $\mathcal{I}$ *satisfies* $\mathcal{T}$ *iff* $C_i^{\mathcal{I}} \subseteq D_i^{\mathcal{I}}$ *holds for all* $C_i \sqsubseteq D_i \in \mathcal{T}$.

| Concepts & Roles | Syntax | Semantics |
|---|---|---|
| atomic concept $\mathbf{C}$ | $A$ | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| atomic role $\mathbf{R}$ | $R$ | $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| inverse role | $R^{-}$ | $\{\langle x, y\rangle \mid \langle y, x\rangle \in R^{\mathcal{I}}\}$ |
| conjunction | $C \sqcap D$ | $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| disjunction | $C \sqcup D$ | $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| negation | $\neg C$ | $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| exists restriction | $\exists R.C$ | $(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$ |
| value restriction | $\forall R.C$ | $(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$ |
| atleast restriction | $\geqslant nS.C$ | $(\geqslant nS.C)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in S^{\mathcal{I}}\} \cap C^{\mathcal{I}}) \geqslant n\}$ |
| atmost restriction | $\leqslant nS.C$ | $(\leqslant nS.C)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in S^{\mathcal{I}}\} \cap C^{\mathcal{I}}) \leqslant n\}$ |
| Role Axioms | Syntax | Semantics |
| role inclusion | $R \sqsubseteq S$ | $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ |
| transitive role | $\mathsf{Trans}(R)$ | $R^{\mathcal{I}} = (R^{+})^{\mathcal{I}}$ |
| role domain | $\mathsf{Domain}(R, C)$ | $\langle x, y\rangle \in R^{\mathcal{I}}$ implies $x \in C^{\mathcal{I}}$ |
| role range | $\mathsf{Range}(R, C)$ | $\langle x, y\rangle \in R^{\mathcal{I}}$ implies $y \in C^{\mathcal{I}}$ |

Figure 1: Syntax and semantics of $\mathcal{SHIQ}$

*A $\mathcal{SHIQ}$-concept $C$ is* satisfiable *w.r.t. a role box $\mathcal{R}$ and a terminology $\mathcal{T}$ iff there is an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$ that satisfies both $\mathcal{R}$ and $\mathcal{T}$. Such an interpretation is called a* model *of $C$ w.r.t. $\mathcal{R}$ and $\mathcal{T}$. A concept $C$ is* subsumed *by a concept $D$ w.r.t. $\mathcal{R}$ and $\mathcal{T}$ iff $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$ for each interpretation $\mathcal{I}$ satisfying $\mathcal{R}$ and $\mathcal{T}$.*

**Theorem 1** *Satisfiability and subsumption of $\mathcal{SHIQ}$-concepts w.r.t. terminologies and role boxes is polynomially reducible to (un)satisfiability of $\mathcal{SHIQ}$-concepts w.r.t. role boxes [9].*

## 3 Tableaux Reasoning with Range and Domain

Here we present an algorithm for deciding the satisfiability of a $\mathcal{SHIQ}$-concept $C$ w.r.t. a role box $\mathcal{R}$; it is an extension of the $\mathcal{SHIQ}$ tableaux algorithm from [9].

For ease of Tableaux construction, we assume $C$ and all concepts in (range and domain axioms in) $\mathcal{R}$ to be in *negation normal form* (NNF), that is, negation occurs only in front of concept names. Any $\mathcal{SHIQ}$-concept can easily be transformed into an equivalent one in NNF by pushing negations inwards; with $\sim C$ we denote the NNF of $\neg C$. We define $\mathsf{RD}(\mathcal{R})$ as the set of concepts s.t. $C \in \mathsf{RD}(\mathcal{R})$ iff $\mathsf{Domain}(R, C) \in \mathcal{R}$ or $\mathsf{Range}(R, C) \in \mathcal{R}$ for some role $R$. We define $\mathtt{cl}(C, \mathcal{R})$ as the smallest set of concepts that is a superset of $C \cup \mathsf{RD}(\mathcal{R})$ and is closed under subconcepts and $\sim$.

**Definition 2** *Let $D$ be a $\mathcal{SHIQ}$-concept in NNF, $\mathcal{R}$ a role box, and $\mathbf{R}_D$ the set of roles occurring in $D$ and $\mathcal{R}$ together with their inverses. Then $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a* tableau *for $D$ w.r.t. $\mathcal{R}$ iff $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \to 2^{\mathtt{cl}(D,\mathcal{R})}$ maps each individual to a set of concepts, $\mathcal{E} : \mathbf{R}_D \to 2^{\mathbf{S}\times\mathbf{S}}$ maps each role to a set of pairs of individuals, and there is some individual $s \in \mathbf{S}$ such that $D \in \mathcal{L}(s)$. Furthermore, for all $s, t \in \mathbf{S}$, $C, C_1, C_2 \in \mathtt{cl}(D, \mathcal{R})$, and $R, S \in \mathbf{R}_D$, it holds that:*

1. *if* $C \in \mathcal{L}(s)$*, then* $\neg C \notin \mathcal{L}(s)$*,*
2. *if* $C_1 \sqcap C_2 \in \mathcal{L}(s)$*, then* $C_1 \in \mathcal{L}(s)$ *and* $C_2 \in \mathcal{L}(s)$*,*
3. *if* $C_1 \sqcup C_2 \in \mathcal{L}(s)$*, then* $C_1 \in \mathcal{L}(s)$ *or* $C_2 \in \mathcal{L}(s)$*,*
4. *if* $\forall S.C \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}(S)$*, then* $C \in \mathcal{L}(t)$*,*
5. *if* $\exists S.C \in \mathcal{L}(s)$*, then there is some* $t \in \mathbf{S}$ *such that* $\langle s, t\rangle \in \mathcal{E}(S)$ *and* $C \in \mathcal{L}(t)$*,*
6. *if* $\forall S.C \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}(R)$ *for some* $R \sqsubseteq^* S$ *with* $\mathsf{Trans}(R)$*, then* $\forall R.C \in \mathcal{L}(t)$*,*
7. $\langle s, t\rangle \in \mathcal{E}(R)$ *iff* $\langle t, s\rangle \in \mathcal{E}(\mathsf{Inv}(R))$*,*
8. *if* $\langle s, t\rangle \in \mathcal{E}(R)$ *and* $R \sqsubseteq^* S$*, then* $\langle s, t\rangle \in \mathcal{E}(S)$*,*
9. *if* $(\leqslant n\ S\ C) \in \mathcal{L}(s)$*, then* $\sharp S^T(s, C) \leqslant n$*,*
10. *if* $(\geqslant n\ S\ C) \in \mathcal{L}(s)$*, then* $\sharp S^T(s, C) \geqslant n$*,*
11. *if* $(\bowtie\ n\ S\ C) \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}(S)$ *then* $C \in \mathcal{L}(t)$ *or* $\sim C \in \mathcal{L}(t)$*,*
12. *if* $\langle s, t\rangle \in \mathcal{E}(S)$ *and* $\mathsf{Domain}(S, C) \in \mathcal{R}$*, then* $C \in \mathcal{L}(s)$*,*
13. *if* $\langle s, t\rangle \in \mathcal{E}(S)$ *and* $\mathsf{Range}(S, C) \in \mathcal{R}$*, then* $C \in \mathcal{L}(t)$*,*

*where we use* $\bowtie$ *as a placeholder for both* $\leqslant$ *and* $\geqslant$ *and we define*

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

**Lemma 1** *A* $\mathcal{SHIQ}$*-concept* $D$ *is satisfiable w.r.t. a role box* $\mathcal{R}$ *iff* $D$ *has a tableau w.r.t.* $\mathcal{R}$*.*

### 3.1 An Extended Tableaux Algorithm

In order to make the following description easier, we will abuse notation by using $\mathsf{Domain}(R)$ and $\mathsf{Range}(R)$ to mean the sets of concepts corresponding to the domain and range axioms in $\mathcal{R}$ that apply to a role $R$, i.e., $\mathsf{Domain}(R) = \{C \mid \mathsf{Domain}(R, C) \in \mathcal{R}\}$, and $\mathsf{Range}(R) = \{C \mid \mathsf{Range}(R, C) \in \mathcal{R}\}$.

**Definition 3** *A* completion tree *for a concept* $D$ *is a tree where each node* $x$ *of the tree is labelled with a set* $\mathcal{L}(x) \subseteq \mathtt{cl}(D, \mathcal{R})$ *and each edge* $\langle x, y\rangle$ *is labelled with a set* $\mathcal{L}(\langle x, y\rangle)$ *of (possibly inverse) roles occurring in* $\mathtt{cl}(D, \mathcal{R})$*; explicit inequalities between nodes of the tree are recorded in a binary relation* $\neq$ *that is implicitly assumed to be symmetric.*

*Given a completion tree, a node* $y$ *is called an* $R$-successor *of a node* $x$ *iff* $y$ *is a successor of* $x$ *and* $S \in \mathcal{L}(\langle x, y\rangle)$ *for some* $S$ *with* $S \sqsubseteq^* R$*. A node* $y$ *is called an* $R$-neighbour *of* $x$ *iff* $y$ *is an* $R$*-successor of* $x$*, or if* $x$ *is an* $\mathsf{Inv}(R)$*-successor of* $y$*. Predecessors and ancestors are defined as usual.*

*A node is* blocked *iff it is directly or indirectly blocked. A node* $x$ *is* directly blocked *iff none of its ancestors are blocked, and it has ancestors* $x'$*,* $y$ *and* $y'$ *such that*

1. $x$ *is a successor of* $x'$ *and* $y$ *is a successor of* $y'$ and
2. $\mathcal{L}(x) = \mathcal{L}(y)$ *and* $\mathcal{L}(x') = \mathcal{L}(y')$ and
3. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

*A node $y$ is* indirectly blocked *iff one of its ancestors is blocked, or it is a successor of a node $x$ and $\mathcal{L}(\langle x, y\rangle) = \emptyset$.*

*For a node $x$, $\mathcal{L}(x)$ is said to contain a* clash *iff $\{A, \neg A\} \subseteq \mathcal{L}(x)$ or if, for some concept $C$, some role $S$, and some $n \in \mathbb{N}$: $(\leqslant n\ S\ C) \in \mathcal{L}(x)$ and there are $n+1$ $S$-neighbours $y_0, \ldots, y_n$ of $x$ such that $C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for all $0 \leq i < j \leq n$. A completion tree is called* clash-free *iff none of its nodes contains a clash; it is called* complete *iff none of the expansion rules is applicable.*

*For a $\mathcal{SHIQ}$-concept $D$, the algorithm starts with a completion tree consisting of a single node $x$ with $\mathcal{L}(x) = \{D\}$ and $\neq = \emptyset$. It applies the expansion rules in Fig. 2, stopping when a clash occurs, and answers "$D$ is satisfiable" iff the completion rules can be applied in such a way that they yield a complete and clash-free completion tree.*

Note that the only change w.r.t. [9] is addition of the *domain* and *range*-rules that add concepts to node labels as required by domain and range axioms.

**Lemma 2** *Let $D$ be an $\mathcal{SHIQ}$-concept.*

1. *The tableaux algorithm terminates when started with $D$.*
2. *If the expansion rules can be applied to $D$ such that they yield a complete and clash-free completion tree, then $D$ has a tableau.*
3. *If $D$ has a tableau, then the expansion rules can be applied to $D$ such that they yield a complete and clash-free completion tree.*

The following theorem is an immediate consequence of Lemmas 1, 2 and Theorem 1.

**Theorem 2** *The tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{SHIQ}$-concepts with respect to role boxes.*

# 4 Role Absorption

Given that the new algorithm is able to deal directly with range and domain axioms, it makes sense to transform GCIs into range and domain axioms. We call this new form of absorption *role absorption* in contrast to the usual form of absorption we will refer to as *concept absorption* (see [10]).

Role absorption is important because in ontology derived KBs range and domain constraints will often have been transformed into GCIs. This is because tools such as OilEd [1] and Protégé [11] are designed to work with range of DL reasoners, some of which (e.g., FaCT) do not support range and domain axioms. Moreover, these forms of GCI are not, in general, amenable to standard concept absorption techniques.

We introduce two kinds of role absorption: *basic* and *extended* role absorptions.

**Basic role absorption.**

The simple form of role absorption, which we will refer to as *basic role absorption*, deals with the axiom of the form $\exists R.\top \sqsubseteq C$ and $\top \sqsubseteq \forall R.C$ and is formalised in the following theorem:

$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-rule: if 1. $\exists S.C \in \mathcal{L}(x)$, $x$ is not blocked, and
2. $x$ has no $S$-neighbour $y$ with $C \in \mathcal{L}(y)$,
then create a new node $y$ with $\mathcal{L}(\langle x, y\rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$

$\forall$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and
2. there is an $S$-neighbour $y$ of $x$ with $C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$

$\forall_+$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and
2. there is some $R$ with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$,
3. there is an $R$-neighbour $y$ of $x$ with $\forall R.C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall R.C\}$

*choose*-rule: if 1. $(\bowtie\ n\ S\ C) \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and
2. there is an $S$-neighbour $y$ of $x$ with $\{C, \sim C\} \cap \mathcal{L}(y) = \emptyset$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \sim C\}$

$\geqslant$-rule: if 1. $(\geqslant n\ S\ C) \in \mathcal{L}(x)$, $x$ is not blocked, and
2. there are not $n$ $S$-neighbours $y_1, \ldots, y_n$ of $x$ with
$C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$
then create $n$ new nodes $y_1, \ldots, y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$,
$\mathcal{L}(y_i) = \{C\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$.

$\leqslant$-rule: if 1. $(\leqslant n\ S\ C) \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and
2. $\sharp S^{\mathbf{T}}(x, C) > n$ and there are two $S$-neighbours $y, z$ of $x$ with
$C \in \mathcal{L}(y), C \in \mathcal{L}(z)$, $y$ is not an ancestor of $x$, and not $y \neq z$
then 1. $\mathcal{L}(z) \longrightarrow \mathcal{L}(z) \cup \mathcal{L}(y)$ and
2. if $z$ is an ancestor of $x$
then $\mathcal{L}(\langle z, x\rangle) \longrightarrow \mathcal{L}(\langle z, x\rangle) \cup \mathsf{Inv}(\mathcal{L}(\langle x, y\rangle))$
else $\mathcal{L}(\langle x, z\rangle) \longrightarrow \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle x, y\rangle)$
3. $\mathcal{L}(\langle x, y\rangle) \longrightarrow \emptyset$
4. Set $u \neq z$ for all $u$ with $u \neq y$

*domain*-rule if 1. $C \in \mathsf{Domain}(S)$, $x$ is not indirectly blocked, and
2. there is an $S$-neighbour $y$ of $x$ and $C \notin \mathcal{L}(x)$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$

*range*-rule if 1. $C \in \mathsf{Range}(S)$, $x$ is not indirectly blocked, and
2. there is an $S$-neighbour $y$ of $x$ with $C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$

Figure 2: The complete tableaux expansion rules for $\mathcal{SHIQ}$

**Theorem 3** *Let $\mathcal{R}$ be a $\mathcal{SHIQ}$ role box.*

1. *An interpretation $\mathcal{I}$ satisfies $\mathcal{R}$ and $\exists R.\top \sqsubseteq C$ iff $\mathcal{I}$ satisfies $\mathcal{R} \cup \{\mathsf{Domain}(R, C)\}$.*

2. *An interpretation $\mathcal{I}$ satisfies $\mathcal{R}$ and $\top \sqsubseteq \forall R.C$ iff $\mathcal{I}$ satisfies $\mathcal{R} \cup \{\mathsf{Range}(R, C)\}$.*

**Extended Role Absorption**

Rewriting techniques similar to those used in concept absorption can be used to extend the basic role absorption technique to deal with a wider range of axioms. An axiom of the form $\exists R.C \sqsubseteq D$ can be absorbed into a domain constraint $\mathsf{Domain}(R, D \sqcup \neg\exists R.C)$ by rewriting it as $\exists R.\top \sqsubseteq D \sqcup \neg\exists R.C$. Similarly, an axiom of the form $D \sqsubseteq \forall R.C$ can be absorbed into a domain constraint $\mathsf{Domain}(R, \neg D \sqcup \neg\exists R.\neg C)$.

# 5 Implementation and Empirical Evaluation

We have implemented the extended tableaux algorithm and role absorption optimisation in the FaCT++ DL reasoner. FaCT++ is a next generation of the well-known FaCT reasoner [7], being developed as part of the EU WonderWeb project (see `http://wonderweb.semanticweb.org/`); it is based on the same tableaux algorithms as the original FaCT, but has a different architecture and is written in C++ instead of Lisp.

**Absorption**

Absorption in FaCT++ uses the same basic approach as FaCT [10, 6]. Given a TBox $\mathcal{T}$, the absorption algorithm constructs a triple of TBoxes $\langle \mathcal{T}_{\text{def}}, \mathcal{T}_{\text{sub}}, \mathcal{T}_{\text{g}} \rangle$ such that:

- $\mathcal{T}_{\text{def}}$ is a set of axioms of the form $A \equiv C$ (equivalent to a pair of axioms $\{A \sqsubseteq C, C \sqsubseteq A\} \subseteq \mathcal{T}$), where $A \in \mathbf{C}$ (i.e., $A$ is a concept name) and there is most one such axiom for each $A \in \mathbf{C}$. Such an axiom is often called a *definition* (of $A$).

- $\mathcal{T}_{\text{sub}}$ consists of a set of axioms of the form $A \sqsubseteq D$, where $A \in \mathbf{C}$ and there is no axiom $A \equiv C$ in $\mathcal{T}_{\text{def}}$.

- $\mathcal{T}_{\text{g}}$ contains all the remaining axioms from $\mathcal{T}$.

The lazy unfolding optimisation allows the axioms in $\mathcal{T}_{\text{def}}$ and $\mathcal{T}_{\text{sub}}$ to be dealt with more efficiently than those in $\mathcal{T}_{\text{g}}$. Therefore, during the absorption process, FaCT++ processes the axioms in $\mathcal{T}_{\text{g}}$ one at a time, trying to absorb them into $\mathcal{T}_{\text{sub}}$. Those axioms that are not absorbed remain in $\mathcal{T}_{\text{g}}$.

To simplify the formulation of the absorption algorithm, each axiom $C \sqsubseteq D$ is viewed as a clause $\mathbf{G} = \{D, \neg C\}$, corresponding to the axiom $\top \sqsubseteq C \rightarrow D$, which is equivalent to $C \sqsubseteq D$. The concepts in $\mathbf{G}$ are also assumed to be in negation normal form. For each such axiom, FaCT++ applies the absorption steps described in Fig. 3, with $\sqcup(\{C_1, \ldots, C_n\})$ being used to denote $C_1 \sqcup \ldots \sqcup C_n$.

In contrast to the FaCT approach, FaCT++ applies all possible simplifications (except recursive absorption) in a single step. This usually leads to several possible concept and role absorption options, with the intention that heuristics will be used to select the "best" absorption. The development of suitable heuristics is, however, still part of future work.

**B** Beginning of the absorption cycle.

**C** Concept absorption. If there is a concept $\neg A \in \mathbf{G}$ such that $A \in \mathbf{C}$ and there is no axiom of the form $A \equiv C$ in $\mathcal{T}_{\mathrm{def}}$, then add $A \sqsubseteq \sqcup(\mathbf{G} \setminus \{\neg A\})$ to $\mathcal{T}_{\mathrm{sub}}$ and exit.

**R** Role absorption. If there is a concept $\neg\exists R.C \in \mathbf{G}$, then add $\mathsf{Domain}(R, \sqcup(\mathbf{G}))$ to $\mathcal{R}$ and exit.

**S** Simplification.

1. For every $C \in \mathbf{G}$ such that $C$ is of the form $(C_1 \sqcup \ldots \sqcup C_n)$, change $\mathbf{G}$ as follows: $\mathbf{G} = \mathbf{G} \cup \{C_1, \ldots, C_n\} \setminus \{C\}\}$.
2. For every $A \in \mathbf{G}$ (resp. $\neg A \in \mathbf{G}$), if there is an axiom $A \equiv C$ in $\mathcal{T}_{\mathrm{def}}$, then substitute $A \in \mathbf{G}$ (resp. $\neg A \in \mathbf{G}$) with $C$ (resp. $\sim C$).
3. If any simplification rule was applied, then return to step **B**.

**E** If there is some $C \in \mathbf{G}$ such that $C$ is of the form $(C_1 \sqcap \ldots \sqcap C_n)$, then for each $C_i$ try to absorb (recursively) $\mathbf{G} \cup \{C_i\} \setminus \{C\}$, and exit. Otherwise, absorption of $\mathbf{G}$ has failed; leave $\mathbf{G}$ in $\mathcal{T}_{\mathrm{g}}$, and exit.

Figure 3: FaCT++ absorption algorithm

**Experiments**

We have tested FaCT++'s performance when classifying several TBoxes derived from realistic ontologies. In each case range and domain constraints from the ontology had already been transformed into GCIs of the form $\exists R.\top \sqsubseteq C$ and $\top \sqsubseteq \forall R.C$ as described above. All tests used FaCT++ version 0.90 beta running under Linux on an Athlon 2000+ machine with 1Gb of memory.

All our experiments shows that classification time and number of operations reduced by approximately 1 order of magnitude after applying basic role absorption, and by a further 60-80% (approximately) after applying extended role absorption (if avaliable). Due to lack of space we only present here results for a single example.

The RTIMS ontology is taken from a publish and subscribe application where it is used by document publishers to annotate documents so that they can be routed to the appropriate subscribers [13]. The ontology contains about 250 concepts (with medium-complex structure), 76 range and domain constraints and 14 GCIs that are not absorbable by concept absorption.

This ontology is too small to show significant gains in performance. In order to give an indication of the effects of extended role absorption on larger Tboxes containing proportionately more GCIs, we duplicating the RTIMS TBox, systematically renaming concepts and roles, and generated larger TBoxes by unioning together several (from 1 to 100) copies of the the original TBox.

The results of our experiments with these Tboxes are shown in Figure 4, with the problem size (number of copies of the original TBox) on the x-axis and classification time in CPU seconds and number of $\sqcup$-rule applications on the y-axis (using a logarithmic scale). It can be seen that without role absorption the classification time (and other y-axis parameters) increases rapidly with problem size, and without extended (basic) role absorption a TBox con-

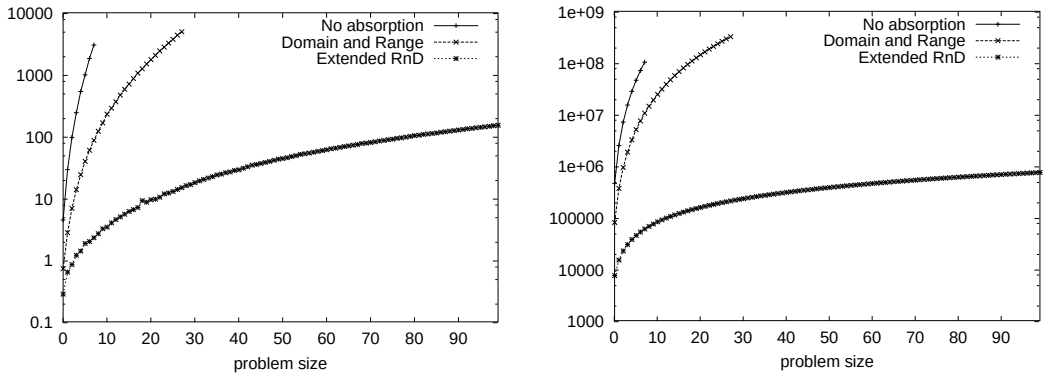



Figure 4: Classification time (left) and $\sqcup$-rule applications (right) for multi-RTIMS TBoxes

sisting of 28 (8) copies of the original already takes several thousand CPU seconds to classify. Further tests failed due to memory limits. In contrast, when using extended role absorption, a TBox consisting of 100 copies of the original could be classified in a little over 100 CPU seconds and requires about 34Mb of memory.

# 6 Discussion

We have shown how a tableaux algorithm for $\mathcal{SHIQ}$ can be extended to support role boxes that include range and domain axioms, and proved that the extended algorithm is still a decision procedure for the satisfiability and subsumption of $\mathcal{SHIQ}$ concepts w.r.t. such a role box. It should be straightforward to similarly extend tableau algorithms for related DLs such as $\mathcal{SHOQ}$. We have also shown how support for range and domian axioms can be exploited in order to add a new form of absorption optimisation called role absorption.

We have implemented the extended algorithm and the role absorption optimisation in the FaCT++ reasoner, and we have illustrated their effectiveness by analysing the behaviour of FaCT++ when classifying several KBs derived from realistic ontologies. The analysis shows that, not only are the new techniques highly effective, but also that the ordering of different absorption steps can have a significant effect on performance. Future work will include a more detailed study of this effect with a view to devising heuristics that can select the most effective absorption for each GCI.

# References

[1] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able 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-Verlag, 2001.

[2] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web on-

tology language 1.0 reference. W3C Proposed Recommendation, 15 December 2003. Available at `http://www.w3.org/TR/owl-ref/`.

[3] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[4] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.

[5] Volker Haarslev and Ralf 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.

[6] I. Horrocks. Implementation and optimisation techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. Cambridge University Press, 2003.

[7] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[8] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, pages 792–797. AAAI Press, 2002.

[9] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei 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.

[10] Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[11] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[12] Pellet OWL reasoner. Maryland Information and Network Dynamics Lab, `http://www.mindswap.org/2003/pellet/`.

[13] Michael Uschold, Peter Clark, Fred Dickey, Casey Fung, Sonia Smith, Stephen Uczekaj Michael Wilke, Sean Bechhofer, and Ian Horrocks. A semantic infosphere. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 882–896. Springer, 2003.

# Mona as a DL Reasoner

Eldar Karabaev and Carsten Lutz
Institute for Theoretical Computer Science
TU Dresden, Germany
{karabaev,lutz}@tcs.inf.tu-dresden.de

**Abstract**

We show how the Mona tool for reasoning in the monadic second order theories WS1S and WS2S can be used to obtain decision procedures for description logics. The performance of this approach is evaluated and compared to the dedicated DL reasoners FaCT and RACER.

## 1 Motivation

The weak monadic second order theories of one and two successors, commonly called WS1S and WS2S, are among the most powerful decidable logics known today [10]. This is witnessed by the fact that a large number of description logics, modal logics, and dynamic logics can be translated into them (see, e.g., [3]), and also by their impressive non-elementary complexity: there exists no positive integer $n$ such that satisfiability of WS1S or WS2S formulas can be decided in $n$-ExpTime [9]. Despite their powerful expressivity and immense computational complexity, with the Mona tool there exists an efficient implementation of both WS1S and WS2S [4]. As the Mona manual puts it, "efficient here means that the tool is fast enough to have been used in a variety of non-trivial settings". Indeed, an impressive list of successful applications can be found on the Mona homepage [1].

Since it is common knowledge that many DLs can be translated into WS2S, it is a natural idea to use Mona for DL reasoning. Thus, *the purpose of the current paper is to translate the basic description logic $\mathcal{ALC}$ into formulas digestible by Mona, and to evaluate Mona's performance for DL reasoning.* More precisely, we use a well-known translation of $\mathcal{ALC}$ into WS2S using a Rabin-style encoding of non-binary trees into binary trees, and exhibit a novel reduction of $\mathcal{ALC}$ into WS1S that is inspired by Pratt's "type elimination" technique for deciding the satisfiability of modal logic formulas. We then compare the performance of Mona with that of the dedicated DL reasoners FaCT and RACER [8, 5]. There are at least two reasons why such a comparison is interesting: first, it contrasts the performance of DL reasoners with the performance of more general reasoners, thus being in the line of [12] where the performance of the FaCT system is compared to that of the first order theorem prover Vampire. Second, Mona implements an automata-based decision procedure, while the two DL reasoners are tableau-based and thus a comparison may contribute to the understanding of the advantages and disadvantages of the two approaches.

The outcome of our investigation is as follows: if no TBoxes are involved, then Mona's performance is reasonable, though it cannot reach the performance of FaCT

and RACER. In the presence of TBoxes, Mona's performance is extremely poor, rendering the Mona approach to DL reasoning virtually useless. However, we believe that using Mona for DL reasoning without TBoxes can be useful at least for prototyping purposes: WS1S and WS2S are powerful enough to accomodate many expressive description logics such as $\mathcal{SHIQ}$ or DLs involving transitive closure of roles, and implementing a translator is considerably less difficult than implementing an optimized, dedicated reasoner. Developing translations for more complex DLs, however, is outside the scope of this paper.

## 2 Translations to WS1S and WS2S

We assume familarity with the description logic $\mathcal{ALC}$, see, e.g., [2] for details. In TBoxes, we admit concept equations $C \doteq D$ with both $C$ and $D$ possibly complex. These TBoxes are interpreted according to the usual descriptive semantics. Due to space limitations, we cannot give a full description of the monadic weak second-order theories WS1S and WS2S. Intuitively, the syntax of our monadic second-order (MSO) language is obtained from the familiar first-order (FO) language without function symbols and constants (but with equality) by

1. restricting predicates to be unary;
2. adding second-order quantifiers "$\forall$" and "$\exists$" that can be used to quantify over unary predicates, which are in this context called second-order variables;
3. in the case of WS1S, adding an ordering predicate "$<$" and a successor function $s(\cdot)$;
4. in the case of WS2S, adding an ordering predicate "$<$" and two successor functions $s_\ell(\cdot)$ and $s_r(\cdot)$.

As for the semantics, formulas of WS1S are interpreted in the structure of one successor function, i.e. in one-side infinite words. The built-in ordering predicate "$<$" has the obvious interpretation on positions in such $\omega$-words, and the successor function can be used for going to the successive position. In the case of WS2S, formulas are interpreted in the structure of two successor functions, i.e. in infinite binary trees. The ordering predicate "$<$" describes the "offspring" relation in such trees, and the two successor functions $s_\ell(\cdot)$ and $s_r(\cdot)$ can be used for going to the left and right successor, respectively. For both WS1S and WS2S, the "W" stands for "weak" indicating that quantification is on *finite* sets rather than on arbitrary ones as in the closely related theories S1S and S2S.

#### To WS1S and the Monadic Theory of Infinite Sets

We first present a translation of $\mathcal{ALC}$ concepts and TBoxes to WS1S, or rather to the MSO theory of infinite sets since we will not use WS1S's built-in ordering predicates and successor functions. The translation is inspired by the Pratt-style "type elimination" procedure for deciding the satisfiability of modal formuas [11]. Intuitively,

type elimination is based on the following simple observation: if a domain element in an $\mathcal{ALC}$ interpretation satisfies a set of concepts $\Gamma$ containing an existential value restriction $\exists R.C$, then there must exist another domain element satisfying $C$ and all $D$ with $\forall R.D \in \Gamma$. This observation also constitutes the core of the WS1S reduction.

Let $C$ be an $\mathcal{ALC}$-concept and $\mathcal{T}$ a TBox. We use $\mathsf{sub}(C)$ to denote the set of subconcepts of $C$ and set

$$\mathsf{sub}(C,\mathcal{T}) := \mathsf{sub}(C) \cup \bigcup_{D \doteq E \in \mathcal{T}} \mathsf{sub}(D) \cup \mathsf{sub}(E).$$

Moreover, we use $\mathsf{cnam}(C,\mathcal{T})$ and $\mathsf{rnam}(C,\mathcal{T})$ to denote the sets of concept names and role names occurring in $C$ and $\mathcal{T}$, respectively. To translate $C$ and $\mathcal{T}$ into a corresponding second order formula, we introduce a unary FO-predicate $P_A$ for each $A \in \mathsf{cnam}(C,\mathcal{T})$, and a unary FO-predicate $Q_D$ for each $D \in \mathsf{sub}(C,\mathcal{T})$ of the form $\exists R.E$ or $\forall R.E$. Then, for each concept $D \in \mathsf{sub}(C,\mathcal{T})$ and each first-order variable $x$, we define a formula $D^\sharp(x)$ by replacing

1. concept names $A$ with $P_A(x)$;
2. $\sqcap$ with $\wedge$ and $\sqcup$ with $\vee$;
3. $\exists R.D$ with $Q_{\exists R.D}(x)$ and $\forall R.D$ with $Q_{\forall R.D}(x)$.

Concerning the last item, we only replace existential and universal value restrictions that are not contained inside another value restriction. For example, taking

$$C = A \sqcap \neg\forall R.(\forall S.(A \sqcap \neg B))$$

and the FO variable $x$, we obtain

$$C^\sharp(x) = P_A(x) \wedge \neg Q_{\forall R.(\forall S.(A \wedge \neg B))}(x).$$

Next, for each role name $R \in \mathsf{rnam}(C,\mathcal{T})$ we define a formula

$$\begin{aligned}\vartheta_R \;:=\; \forall x.\exists W.\Big[ & \bigwedge_{\forall R.C \in \mathsf{sub}(X,\mathcal{T})} \Big(Q_{\forall R.C}(x) \to \big(\forall y.W(y) \to C^\sharp(y)\big)\Big) \\ & \wedge \bigwedge_{\exists R.C \in \mathsf{sub}(X,\mathcal{T})} \Big(Q_{\exists R.C}(x) \to \big(\exists y.W(y) \wedge C^\sharp(y)\big)\Big)\Big]\end{aligned}$$

where $x$ is a first order variable and $W$ is a second order variable (the same variables may be used for every $R \in \mathsf{rnam}(C,\mathcal{T})$). Finally, we can define the WS1S formula $\varphi^1_{C,\mathcal{T}}$, which is the translation of $C$ and $\mathcal{T}$:

$$\varphi^1_{C,\mathcal{T}} := \exists x.C^\sharp(x) \wedge \forall x.\Big[\bigwedge_{D \doteq E \in \mathcal{T}} (D^\sharp(x) \leftrightarrow E^\sharp(x))\Big] \wedge \bigwedge_{R \in \mathsf{rnam}(C,\mathcal{T})} \vartheta_R$$

Some remarks on this translation, which is called $\mathsf{T1}$ in the remainder of this paper, are in order. First, it is linear and may even result in an exponential compression of

the input concept and TBox. Second, the formula $\varphi^1_{C,\mathcal{T}}$ does not refer to any successor functions and ordering predicates, and can hence be interpreted both in WS1S and in the MSO theory of infinite sets. And third, it is interesting to note that description logic roles are not explicitly represented on the second-order side. This is so because we have only a single relation available in WS1S: the successor function, which does not seem suitable for representing the in general non-functional role relationships. Thus we resort to a type elimination perspective as initially sketched: the existentially quantified variable $W$ in $\vartheta_R$ comprises those domain elements that are needed to satisfy all the existential value restrictions of $x$.

**Theorem 1.** *A concept $C$ is satisfiable w.r.t. a TBox $\mathcal{T}$ iff $\varphi^1_{C,\mathcal{T}}$ is satisfiable in the MSO-theory of infinite sets iff $\varphi^1_{C,\mathcal{T}}$ is WS1S-satisfiable.*

It is interesting to note that we can even eliminate the second-order quantifier in the previous translation by defining the formula $\vartheta_R$ in a different way. Here, the relation to type elimination is even more visible:

$$\vartheta'_R := \forall x. \bigwedge_{\forall R.C \in \mathsf{sub}(X,\mathcal{T})} \Big[ Q_{\exists R.C}(x) \to \Big( \exists y. \big( C^\sharp(y) \wedge \bigwedge_{\forall R.D \in \mathsf{sub}(X,\mathcal{T})} (Q_{\forall R.D}(x) \to D^\sharp(y)) \big) \Big) \Big]$$

With this modification, we could use our translation together with a first-order theorem prover rather than together with Mona—a path that we are not going to explore in the current paper. Note that the modified translation is quadratic rather than linear.

### To WS2S

The translation to WS2S is much more standard than our WS1S translation. Since we have two successor functions available in WS2S, we can now explicitly represent the relational structure on the second-order side: $\mathcal{ALC}$ is known to have the tree-model property, and an old trick of Rabin [10] can be used to transform $\mathcal{ALC}$'s tree models, which are not necessarily binary trees, into the binary tree structure of WS2S.

Let $C$ be an $\mathcal{ALC}$ concept and $\mathcal{T}$ a TBox. For the new translation, we fix an enumeration of the roles in $C$ and $\mathcal{T}$, and use $\#R$ to denote the position of a role $R$ in this enumeration. We introduce a unary FO-predicate $P_A$ for each $A \in \mathsf{cnam}(C,\mathcal{T})$, and another unary FO-predicate $M$. Then, we inductively define two translation functions $\cdot^x$ and $\cdot^y$, where $x$ and $y$ are first-order variables. Here is the $\cdot^x$ translation:

$$\begin{array}{rcll}
A^x & := & A(x) & \\
(\neg C)^x & := & \neg C^x & \\
(C \sqcap D)^x & := & C^x \wedge D^x & \\
(C \sqcup D)^x & := & C^x \vee D^x & \\
(\exists R.C)^x & := & \exists y.(y <_\ell s_r^n(x) \wedge C^y \wedge M(y)) & \text{where } n := \#R \\
(\forall R.C)^x & := & \forall y.((y <_\ell s_r^n(x) \wedge M(y)) \to C^y) & \text{where } n := \#R
\end{array}$$

In this translation, $s_r(\cdot)$ is the "right" successor function, and $s_r^n(\cdot)$ stands for going to the right successor $n$ times. Moreover, $<_\ell$ is the ordering over left successors only.

Since this is not available in Mona, we simulate $y <_\ell x$ by writing

$$\exists Q.\Big[x \in Q \wedge \forall z.\big(z \in Q \rightarrow (z = y \vee s_\ell(z) \in Q)\big)\Big].$$

Note that this formula only works since, in WS2S, quantification is over *finite* sets. The $\cdot^y$ translation is defined symmetrically to $\cdot^x$, details are omitted. Given a concept $C$ and a TBox $\mathcal{T}$, we now define their translation to WS2S as follows:

$$\varphi^2_{C,\mathcal{T}} := \exists x.(M(x) \wedge C^x) \wedge \bigwedge_{D \doteq E \in \mathcal{T}} \forall x.(D^x \leftrightarrow E^x).$$

The intuition behind this translation, which is called T2, is as follows: there is a one-to-one correspondence between domain elements of $\mathcal{ALC}$ interpretations and nodes in the WS2S tree structure that are in the extension of the predicate $M$. Let $x$ be a node in the WS2S tree that is in $M$. To find the $R$-successors of $x$ for a role name $R \in \mathsf{rnam}(C, \mathcal{T})$ with $\sharp R = n$, we start at $x$ and follow the right successor function exactly $n$ times to the node $y$. Then the $R$-successors of $x$ are $y$, its left successor, its left successor's left successor, and so forth. This also explains the use of the $M$ predicate: since we do not want to have infinitely many successors for each domain element and each role name, we mark the "existing ones" with $M$.

**Theorem 2.** *A concept $C$ is satisfiable w.r.t. a TBox $\mathcal{T}$ iff $\varphi^2_{C,\mathcal{T}}$ is WS2S-satisfiable.*

There are some interesting variations of this translation. For example, we can modify the translation $\cdot^x$ as follows (and $\cdot^y$ analogously):

$$\begin{array}{lcll} (\exists R.C)^x & := & M(s^n_r(x)) \wedge \exists y.(y <_\ell s_\ell(s^n_r(x)) \wedge C^y) & \text{where } n := \#R \\ (\forall R.C)^x & := & M(s^n_r(x)) \rightarrow \forall y.(y <_\ell s_\ell(s^n_r(x)) \rightarrow C^y) & \text{where } n := \#R \end{array}$$

Here, the intuition of the $M$ predicate is a different one since $M$ is only used to state whether a node $x$ has successors for a role name $R$ at all: this is the case if and only if $s^n_r(x)$ is in $M$, with $n = \sharp R$. This second variant of the S2S translation is denoted with T2b.

## 3 Evaluation

To evaluate the performance of Mona when used for deciding the satisfiability of $\mathcal{ALC}$ concepts without reference to TBoxes, we employed two different classes of concepts. Firstly, we tested our approach on the Tableaux'98 (henceforth T98) benchmark suite that is frequently used to evaluate DL reasoners, see, e.g., [8, 12]. The T98 suite consists of 18 sequences of concepts, each sequence comprised of 21 concepts with increasing difficulty. Since the T98 concepts are artificial in the sense that they have been designed with the only purpose of making reasoning difficult [6], we secondly used concepts that were extracted from the real world knowledge bases Galen and Platt—see [7] for more information on both of them.

All tests were perfomed five times: once for each of the three translations T1, T2, and T2b, and once using the well-known DL reasoners FaCT and RACER [8, 5]. The

| | T1 | T2 | T2b | RACER | FaCT |
|---|---|---|---|---|---|
| k_branch_n | 0 | 2 | 1 | 14 | 6 |
| k_branch_p | 0 | 2 | 1 | 20 | 8 |
| k_d4_n | 0 | 2 | 4 | 21 | 21 |
| k_d4_p | 0 | 3 | 6 | 21 | 21 |
| k_dum_n | 0 | 2 | 3 | 21 | 21 |
| k_dum_p | 0 | 6 | 7 | 21 | 21 |
| k_grz_n | 0 | 3 | 5 | 21 | 21 |
| k_grz_p | 0 | 4 | 5 | 21 | 21 |
| k_lin_n | 1 | 1 | 21 | 21 | 21 |
| k_lin_p | 1 | 7 | 10 | 21 | 21 |
| k_path_n | 0 | 3 | 16 | 21 | 8 |
| k_path_p | 0 | 4 | 17 | 21 | 10 |
| k_ph_n | 2 | 4 | 11 | 21 | 9 |
| k_ph_p | 2 | 4 | 11 | 9 | 8 |
| k_poly_n | 0 | 1 | 2 | 21 | 21 |
| k_poly_p | 0 | 1 | 1 | 21 | 21 |
| k_t4p_n | 0 | 0 | 6 | 21 | 21 |
| k_t4p_p | 0 | 1 | 10 | 21 | 21 |
| Total: | 6 | 50 | 137 | 358 | 301 |
| Galen-kris-1,% | 97.34 | 93.13 | 91.07 | 100.00 | 100.00 |
| Galen-kris-2,% | 99.54 | 98.66 | 97.83 | 100.00 | 100.00 |
| Platt,% | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |

Figure 1: Experimental Results

results are summarized in Figure 1. In the table, the names k_*_* denote the concept sequences of T98. The entries are to be read as follows: the reasoners had 100 seconds to decide the satisfiability of each concept in a sequence. The number given in the table is then the number of the last concept that a reasoner was able to solve within this time. Thus, the entry "21" means that all concepts in a given sequence have been solved. For the "real world" concepts, for which the results can be found in the last three lines, we use a different scheme: there are again 100 seconds available for reasoning on each concept, but since these concepts are not ordered w.r.t. increasing difficulty, we simply give the percentage of concept that the reasoner was able to solve within the given time. Note that, in total, there were 1165 concepts for Galen-kris-1, 1936 concepts for Galen-kris-2, and 262 concepts for Platt

There are several interesting observations to be made in Figure 1. First, it is obvious that the dedicated DL reasoners RACER and FaCT outperform Mona in most cases, except for the translation T2b used on the sequences k_lin_n, k_path_p, k_path_n, k_ph_n, and k_ph_p—in the last case, Mona even beats FaCT *and* RACER. Second, on the T98 concepts the translation T2b performs much better than the other two translations, and T1 is worst solving only 6 concepts out of 378. Surprisingly, the situation is reversed for the real world concepts: here T1 outperforms both other translations and T2b is worst. And third, there is a surprising difference in the performance of the almost identical T2 and T2b translations (at least on the T98 concepts). Thus, Mona is apparently quite sensitive to small changes in the translation.

How should these results be judged? Let us start with saying that the better performance of FaCT and RACER is perhaps not too surprising: both Mona and the involved DL reasoners are highly optimized, but Mona is capable of dealing with a much more powerful logic. This is witnessed by the fact that the complexity of WS1S and WS2S is non-elementary [9], while FaCT and RACER implement ExpTime-complete logics. Still, we believe that the performance of our translations is reasonable. In particular, it should be taken into account that, compared to the implementation of a full-fledged DL reasoner, the implementation of our translations is a piece of cake. Since many different description logics can be translated to WS1S and WS2S, we believe that such translations can be useful at least for prototyping purposes.

To analyze why the three translations exhibit a different performance, we need to introduce some of Mona's implementation details. To decide the satisfiability of a formula, Mona constructs a finite automaton that accepts the empty language if and only if the input formula is unsatisfiable, and then performs an emptyness test on this automaton. The construction of the automaton, which works on $\omega$-words in the case of WS1S and on infinite trees in the WS2S case, involves a number of automata-theoretic operations: complementation for dealing with logical negation, union for disjunction, product for conjunction, and projection for quantifiers. Since Mona always works with *determinstic* automata, the most "dangerous" operation in this list is projection as it involves the determinization of a non-deterministic automaton. As is well-known, this may produce an exponential blowup in automaton size, in contrast to constant blowup produced by complementation and union, and quadratic blowup produced by the product. This is closely related to the fact that WS1S and WS2S are non-elementary: a separate projection cannot be avoided for each alternation of logical quantifiers, thus resulting in repeated exponential blowups.

Surprisingly, however, quantifier alternation is almost never the culprit of performance problems in our translation-based appoach to DL reasoning: the reason for non-termination is usually the more harmless looking product operation, i.e. the treatment of logical conjunction. To understand this, we must know some more details about Mona internals: the alphabet of the automaton constructed for a (sub)formula $\varphi$ is comprised of 0/1-strings whose length is identical to the number of free variables in $\varphi$—the intuition is that the domain element described by such a string is contained in the extension of exactly those free variables for which we find a "1" value in the string. Thus, the size of the alphabet is exponential in the length of the input formula. To cope with this, Mona stores the transition table of automata in a compressed way using binary decision diagrams (BDDs). Unfortunately, this compression only works well if the underlying formula enforces many interdependencies among the free variables.

Now consider our translation T1: each subformula $\vartheta_R$ is a huge conjunction inside a "$\forall\exists$" quantifier pattern. Moreover, each conjunct involves a large number of free variables: since first-order predicates are treated by Mona as free variables (which are implicitly existentially quantified on the outermost level), this includes the predicates $P_A$ introduced for concept names and the predicates $Q_C$ introduced for subconcepts $C$ of the form $\exists R.D$ and $\forall R.D$. Unfortunately, the interaction between these free variables turns out to be rather weak in general. Thus, the repeated product operations

performed when processing the huge conjunction in $\vartheta_R$ produce an automaton whose BDD has an excessive number of nodes. The subsequent projection and determinization that is performed to treat the second operator in the "$\forall\exists$" pattern is not able to process this large automaton.

In contrast, the formulas produced by T2 and T2b reflect the structure of the original concept, and are thus not just a big conjunction. This explains why T2b is better on the T98 benchmark. But why, then, is T1 better on real world concepts? There seem to be two reasons: first, the concepts extracted from real world knowledge bases are much smaller than the T98 ones, which can be several megabytes in size. Thus, conjunctions produced by T1 are less gigantic in the real world case. Second, Mona can handle automata on $\omega$-words more efficiently than tree-automata, thus giving T1 an advantage over T2 and T2b.

We have also used our translation for deciding the satisfiability of concepts w.r.t. TBoxes, e.g. on the Platt knowledge base and on the two $\mathcal{ALC}$ versions of Galen. Unfortunately, the results were discouraging: Mona terminated only for very small TBoxes ($< 10$ concept equations) and was not able to classify any of the real world KBs. The reason for this is again due to the generation of BDDs with a massive number of nodes: in all our translations, TBoxes are translated into a huge conjunction inside a "$\forall$" quantifier. Hence, we can observe the same blowup pattern as with T1 on the T98 concepts. Even a preceeding "absorption" of concept equations as known from [7], and an elimination of "non-relevant" concept equations as proposed in [12] did not allow us to classify real world KBs.

## 4 Discussion

We have shown how the Mona tool can be exploited for reasoning about description logics. The outcome of our experiments suggests that, although Mona is outperformed by dedicated DL reasoners, her performance is sufficient at least for prototyping purposes. In this context, it should be noted that translations can be implemented rather quickly, and that Mona is expressive enough to capture a large class of description logics. For example, it should not be hard to come up with translations for more powerful DLs such as $\mathcal{SHIQ}$, and even to treat features that are very difficult for tableau reasoners, such as the transitive closure of roles.

It would be very interesting to determine a class of concepts on which Mona performs good, but the DL reasoners do not, and vice versa. A first idea is provided by the discussion in Section 3: Mona is good on disjunction (the corresponding operation "automata union" only yields a constant blowup in size), and bad on conjunction; for tableau algorithms, this is the other way round. Still, we found it difficult to come up with the desired class of concepts due to the intricate optimization techniques of FaCT and RACER. The two reasoners even behave quite differently: we have found two classes of formulas on which Mona performs good, but one of FaCT and RACER performs bad. A class of such formulas on which *both* FaCT and RACER perform bad, however, remains yet to be seen.

The translator and concepts used in the experiments can be downloaded from the

internet, c.f. http://lat.inf.tu-dresden.de/~clu/atom.tar.gz.

## References


[1] The Mona homepage. http://www.brics.dk/mona/.

[2] F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2003.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[4] J. Elgaard, N. Klarlund, and A. Moller. Mona 1.x: new techniques for ws1s and ws2s. In *Computer Aided Verification, CAV '98, Proceedings*, volume 1427 of *LNCS*. Springer Verlag, 1998.

[5] V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 701–705. Springer-Verlag, 2001.

[6] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Technical Report IAM-96-015, University of Bern, Switzerland, 1996.

[7] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. Phd thesis, University of Manchester, 1997.

[8] I. Horrocks. Using an expressive description logic: Fact or fiction? In *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR98)*, pages 636–647, 1998.

[9] A. Meyer. Weak monadic second order theory of successor is not elementary-recursive. In *LOGCOLLOQ: Logic Colloquium*. Lecture Notes in Mathematics, No. 453, Springer-Verlag, 1975.

[10] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[11] V. R. Pratt. Models of program logics. In *Proceedings of the Twentieth Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979.

[12] D. Tsarkov and I. Horrock. Dl reasoner vs. first-order prover. 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/), pages 152–159, 2003.

# OntoXpl*
# Exploration of OWL Ontologies

**Volker Haarslev and Ying Lu and Nematollah Shiri**
Computer Science Department
Concordia University, Montreal, Canada
haarslev@cs.concordia.ca
ying_lu@cs.concordia.ca
shiri@cs.concordia.ca

**Abstract**

This paper describes the OWL ontology explorer OntoXpl. It is available as a web server based on the tomcat architecture. Standard HTML browsers can be used to interact with OntoXpl. At least three potential user groups are targeted by OntoXpl's design: (i) users with a limited background of ontologies and OWL; (ii) ontology developers that are OWL experts; (iii) users interested in understanding and reusing existing ontologies. OntoXpl is intended to complement existing ontology editors and does not offer any editing support. The current implementation of OntoXpl is based on the OWL DL reasoner Racer and uses Racer's extensive query interface in order to support the exploration of OWL ontologies.

## 1 Introduction

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research. In particular, the semantic web initiative [3] is based on description logics (DLs) and defines important challenges for current system implementations. Recently, one of the main standards for the semantic web has been proposed: the Web Ontology Language (OWL) [17]. OWL is based on two other standards: Resource Description Format (RDF [10]) and its corresponding "vocabulary language" RDF Schema (RDFS) [4]. In recent research efforts, these languages are mainly considered as ontology representation languages (see e.g. [1] for an overview). The languages are used for defining classes of so-called abstract objects. Now, many applications start to use the RDF part of OWL for representing information about specific abstract objects of a certain domain. Graphical editors such as OilEd [2] or Protégé [14] support this way of using OWL quite well.

State-of-the-art description logic (DL) inference systems such as Racer allow for interpreting OWL ontology documents as T-boxes and A-boxes [8]. Racer accepts the OWL DL subset [17] (with the additional restriction of approximated reasoning

*OntoXpl's download page: http://www.cs.concordia.ca/ying_lu/

for so-called nominals and no full number restrictions for datatype properties). Descriptions of individuals are represented as A-boxes by the RACER System (for details see the RACER User's Guide [7]). Viewing the RDF part of OWL DL documents as A-boxes provides for query languages supported by DL systems. Furthermore, graphical interfaces for description logic inference systems can be used to inspect OWL ontologies.

User interfaces are very important for practical work with description logic inference systems. An increasing number of graphical interfaces are available for existing DL systems. One class of interfaces consists of ontology editors such as OILED [2] and PROTÉGÉ [14]. With these editors ontologies can be interactively built and they can be stored, for example, as OWL documents. In addition, the editors can be used to develop RDF documents for describing information about individuals with respect to OWL ontologies. Applications using these OWL documents require an inference engine that supports reasoning about individuals. Indeed, OILED and PROTÉGÉ can be configured to use RACER [6] as an inference engine for classifying ontologies and for answering simple queries about individuals.

The second class of interfaces offers browsing and visualization capabilities. RICE [13] supports the input of textual queries and displays the concept/class hierarchy of T-boxes as outline views as well as the relational structure of A-boxes as directed graphs. The outline view of classes is usually also supported by ontology editors but RICE additionally supports the visualization of A-boxes. Other OWL/RDF visualization tools or editors with visualization capabilities are, e.g., KAON [15], OntoEdit [16], and OntoTrack [11].

The OWL ontology explorer ONTOXPL presented in this paper is intended to complement existing ontology editors and visualization tools. It is completely based on OWL and offers a large variety of information queries. Three potential user groups are targeted by ONTOXPL's design: (i) users with a limited background of ontologies and OWL; (ii) ontology developers that are OWL experts; (iii) users interested in understanding and reusing existing ontologies. ONTOXPL is available as a web server based on the tomcat architecture. Standard HTML browsers can be used to interact with ONTOXPL. Its interface makes heavy use of RACER's extensive query interface in order to support users when exploring OWL ontologies. The following sections give a brief tour on using ONTOXPL and explain its rationale in more detail. Afterwards ONTOXPL is compared with related work. This paper concludes with an outlook to possible future work.

## 2 OntoXpl's main user interface

ONTOXPL's design is influenced by OWL (and its foundation on DLs).[1] Therefore, it focuses on the three main language elements of OWL, classes/concepts, roles/properties, and nominals/individuals.

The main command pane of ONTOXPL is shown in Figure 1. The filename of the OWL ontology currently loaded into ONTOXPL and RACER is shown with a summary of the number of contained concept and role names (see also Section 3 for an

[1] The DL and OWL vocabulary is used interchangeably in this paper.

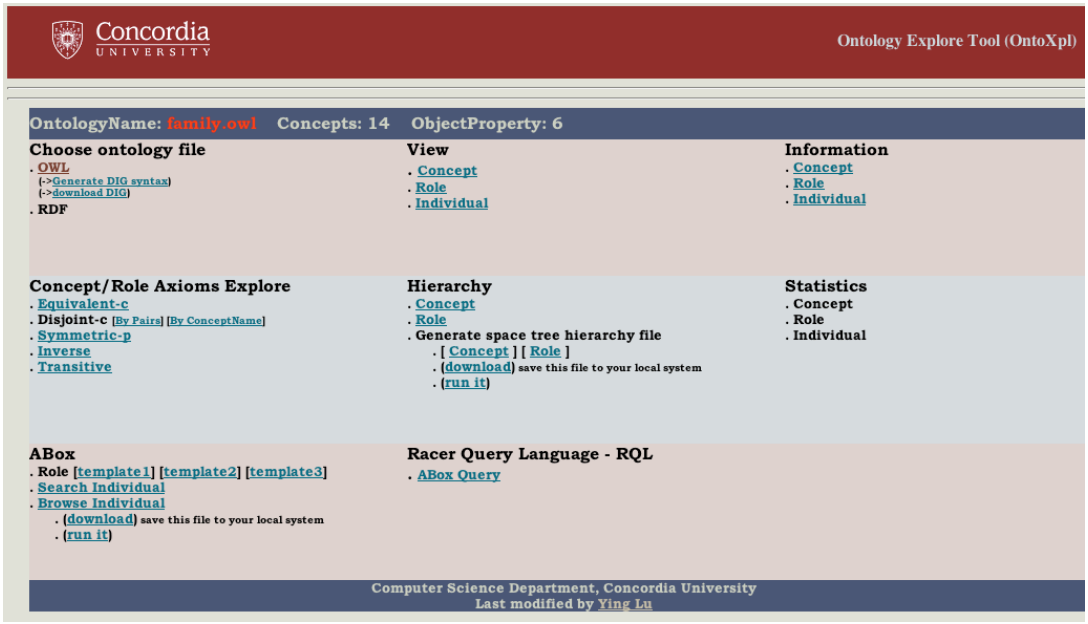



Figure 1: OntoXpl's main command pane.

explanation of the example knowledge base). OntoXpl's interface offers eight principal browsing categories (from left to right and top to bottom): file selector, "natural language" description, structural information, exploration of concept/property axioms, inspection of concept and role hierarchies, view of statistical information (not yet implemented), inspection of A-box graph structures, and the interactive use of Racer's query language nRQL. In the following the seven implemented categories are described.

Figure 2 shows a zoom of the first (horizontal) command pane. The left group of commands is used to load an OWL file and generate a DIG representation of the loaded OWL file. The middle group of commands applies to concepts, roles, and individuals. These commands result in displaying the OWL source code (e.g., see Figure 8) together with a "natural language" description (e.g., see Figure 7). The "natural language" (NL) description is based on the DL notation and tries to describe the selected item w.r.t. this notation. These NL descriptions are intended for users with a limited background on DL and OWL. The information views of concepts (e.g., see the window at the left-hand side of Figure 9), roles (e.g., see Figure 10), and individuals (e.g., see the two windows at the right-hand side of Figure 9) use Racer's query interface to display their (inferred) characteristics. Concepts are described by (i) their relative position in the classification hierarchy (e.g., parent, children), (ii) the roles occurring in the concept declarations, and (iii) the individuals that are instances of this concept. By analogy, a role is similarly described but in addition to its position in the role hierarchy, the concepts are listed that use this role. An individual is described by (i) its most specific concept names (so-called types) of which it is an


Figure 2: Zoom of the upper three command menus (from left to right): file selection, OWL / natural language views, information page views.

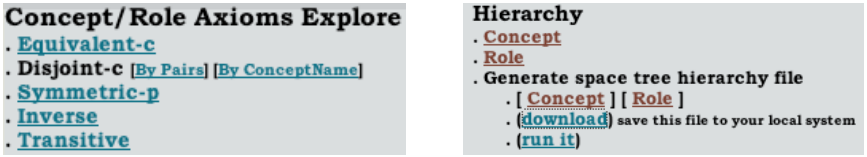



Figure 3: Zoom of the middle two command menus (from left to right): explore concept/property characteristics, show concept/role hierarchies.

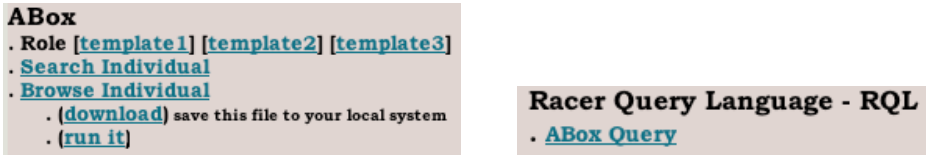



Figure 4: Zoom of the bottom two command menus (from left to right): A-box command menu, nRQL Racer Query Language.

instance, (ii) other individuals that are instances of concepts (parents, children, etc) related to its types.

The two implemented command groups from the middle pane are shown in Figure 3. The command group displayed on the left allows one to query about equivalent or disjoint concept names and symmetric, inverse, and transitive roles. The other group is concerned with concept and role hierarchies. There exist two principal services: (i) one can browse the concept or roles hierarchies in an outline view; (ii) a data file for the SpaceTree tool [5] is generated such that the taxonomies can be graphically inspected.[2]

The last two command groups from the bottom pane are shown in Figure 4. They are dedicated to explore A-boxes. The first command group has several search forms to retrieve individuals and their known relationships with other individuals, to browse relationships in an outline view or inspect the A-box structure with SpaceTree. The second command group allows users to query A-boxes with Racer's query language nRQL [9].

[2]The hierarchy is shown as a pure tree, i.e., edges to more than one superclass are ignored.

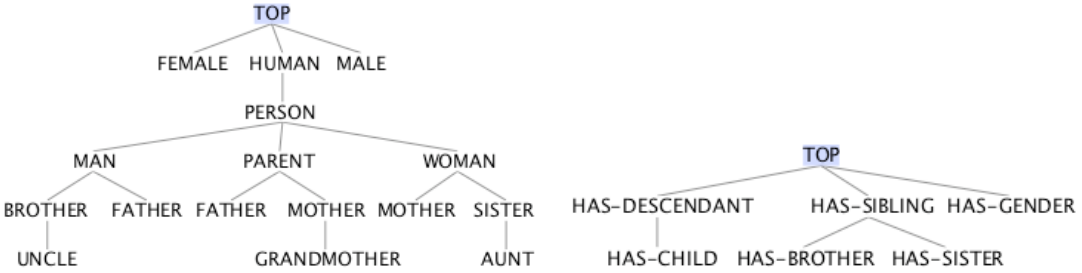



Figure 5: Class (left) and property hierarchy (right) of the "family" KB.

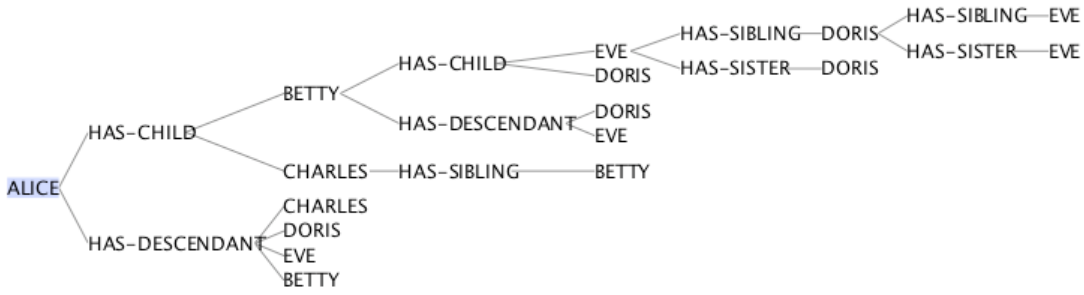



Figure 6: Graph of the A-box relationships.

```
It is the anonymous subclass of
   A concept HUMAN
   and
   it has a filler in the role HAS-GENDER
   at least one (or more than one) of its instances is(are):
   A concept FEMALE
         or
   A concept MALE
```

Figure 7: "Natural Language" description of class PERSON.

## 3 Example scenario

The capabilities of OntoXpl are best explored interactively. However, in this section we try to briefly illustrate some of its main features. Let us assume that OntoXpl is used to explore an ontology file called "family.owl" describing knowledge about family members (e.g., mother, aunt) and their relationships (e.g., has-child, has-sibling). The structure of the corresponding class and role hierarchies is shown in Figure 5 and the structure of the A-box in Figure 6. From the T-box graph a user might be interested in the class PERSON and selects this class for further inspection. Figure 7 shows a "natural language" (NL) description of this class (the underlined names link to the

```
(rdfs:subClassOf)
        (owl:Class)
                (owl:intersectionOf rdf:parseType="Collection")
                        (owl:Class rdf:about="HUMAN")
                        (/owl:Class)
                        (owl:Restriction)
                                (owl:onProperty rdf:resource="HAS-GENDER")
                                (/owl:onProperty)
                                (owl:someValuesFrom)
                                        (owl:Class)
                                                (owl:unionOf rdf:parseType="Collection")
                                                        (owl:Class rdf:about="FEMALE")
                                                        (/owl:Class)
                                                        (owl:Class rdf:about="MALE")
                                                        (/owl:Class)
                                                (/owl:unionOf)
                                        (/owl:Class)
                                (/owl:someValuesFrom)
                        (/owl:Restriction)
                (/owl:intersectionOf)
        (/owl:Class)
(/rdfs:subClassOf)
```

Figure 8: OWL specification of class PERSON.

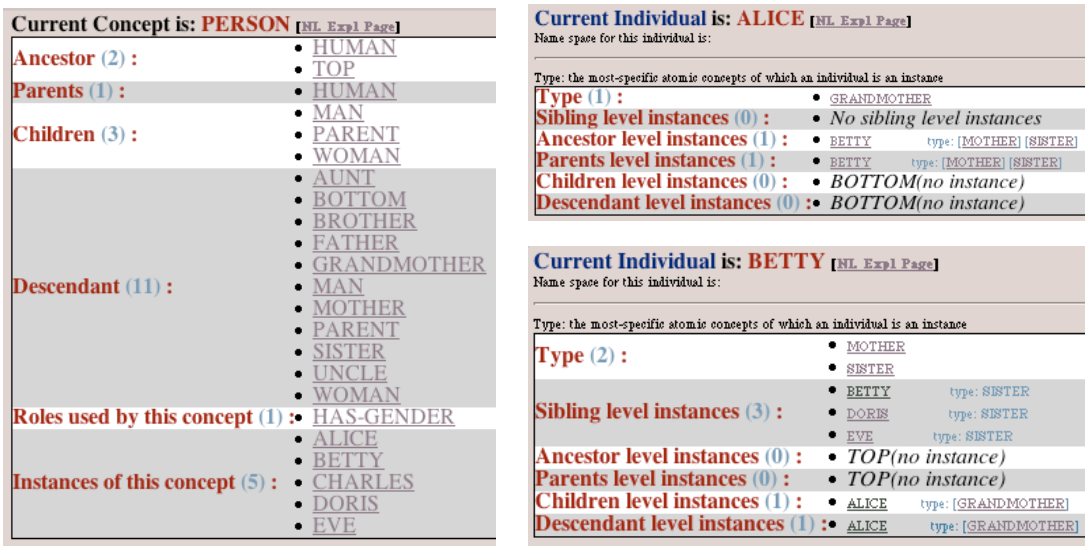



Figure 9: Taxonomic information about the class PERSON (left) and the individuals ALICE (top-right) and BETTY (bottom-right).

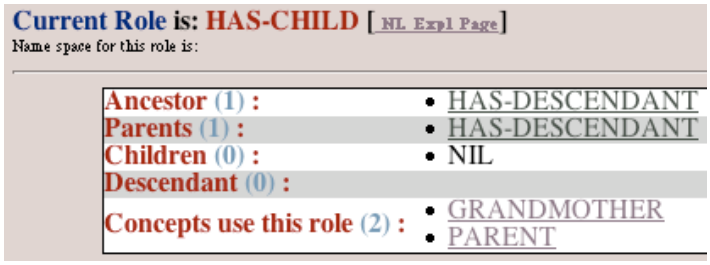



Figure 10: Taxonomic information about the role HAS-CHILD.

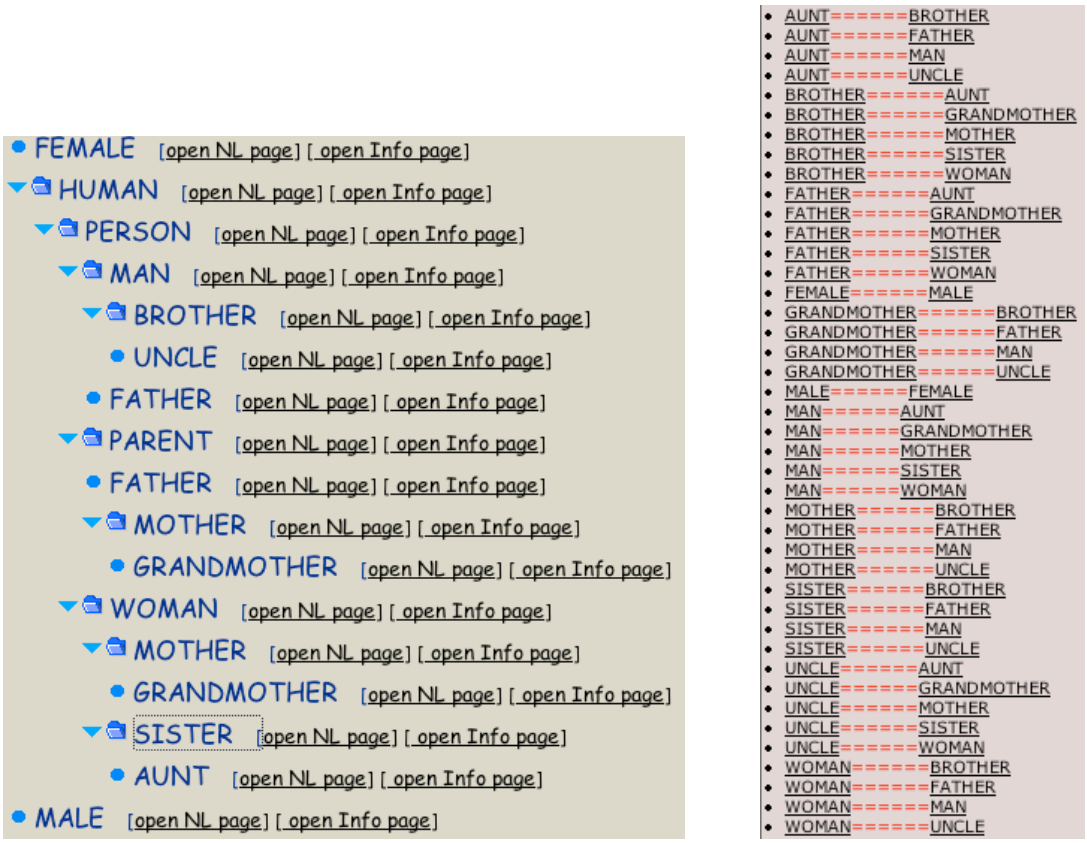



Figure 11: Outline view of class hierarchy (left) and all pairs of disjoint concept names (right).

corresponding NL views) while Figure 8 displays its OWL specification (using the XML syntax). The NL and OWL views are directly linked with the corresponding taxonomic views. Figure 9 displays the taxonomic information about the class PERSON retrieved from RACER. It lists ancestors, parents, children, and descendants of PERSON. It also shows the role names used in this class specification and the individuals which are instances of PERSON.

A user might be interested in the individual ALICE. Its taxonomic information is shown in Figure 9, e.g., ALICE is an instance of GRANDMOTHER. This view also lists instances of concepts that are ancestors, parents, siblings, descendants, or children of ALICE's most-specific subsumers (GRANDMOTHER). For instance, BETTY is an instance of these parent classes. The corresponding information about BETTY is shown in Figure 9. Figure 10 shows the taxonomic information about the role HAS-CHILD. The display of inferred information in these windows is intended to help users better understand the structure of the T-boxes and A-boxes.

In contrast to the hierarchical views displayed by using the SpaceTree tool, ONTOXPL also offers its own outline views for concept and role hierarchies as well as

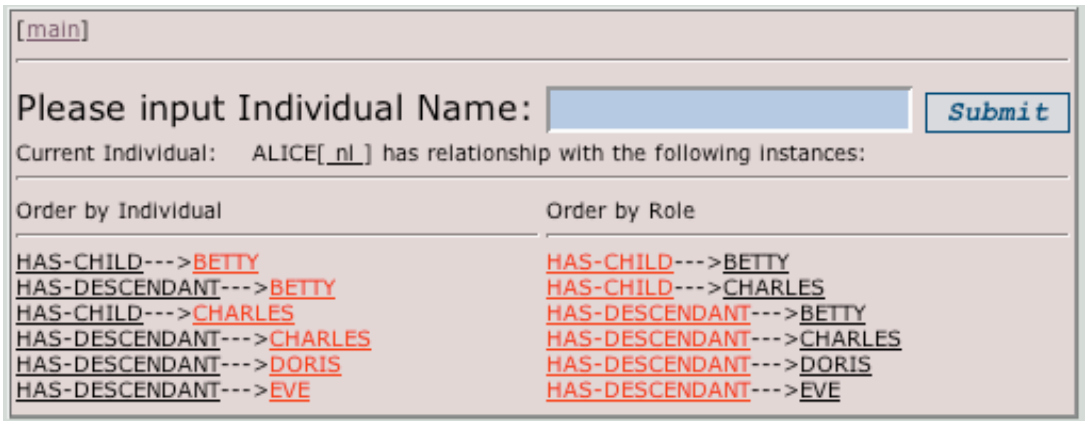



Figure 12: Asserted and inferred role fillers of ALICE.

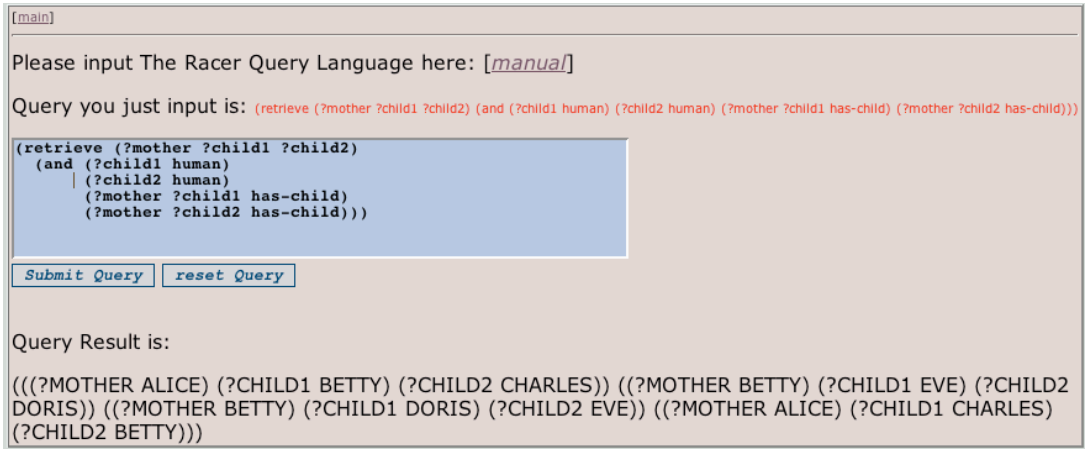



Figure 13: Example nRQL query and its result.

A-box structures. The left-hand side of Figure 11 shows the complete unfolded hierarchy using an outline view. The disadvantage of this type of view is the repeated occurrence of classes (or subtrees) that have more than one parent (e.g., FATHER, MOTHER). The right-hand side of Figure 11 displays all pairs of disjoint concept names.

Figure 12 displays information about the asserted and inferred role fillers of ALICE (ordered by individual or role name). Figure 13 shows a complex nRQL [9] query which searches for children having a common mother. The dialog box displays the input query and its returned result in a Lisp-like notation.

## 4 Discussion

Currently there do not exist many stable and usable ontology visualization or exploration tools (and even editors). The lack of suitable tools and their shortcomings were one of the major motivations to design and implement OntoXpl. The motivation for OntoXpl's web server based architecture was the ease of use with standard HTML browsers and the simple adaptation to multi-user environments. To the best of our knowledge OntoXpl is currently the only ontology exploration tool that is fully targeted to OWL and relies on Racer's deductive capabilities for offering users better exploration capabilities. A detailed description of OntoXpl and its architecture as well as a comparison with related work can be found in [12].

Various features of OntoXpl are also (partly) supported by ontology editors such as Protégé [14] and OilEd [2] or OWL/RDF visualization tools or editors with visualization capabilities such as KAON [15], OntoEdit [16], and OntoTrack [11]. For instance, Protégé also offers users a high-level (DL-like) description of the definition of concept names if the mouse pointer is moved over these names. However, this does not seem to be very suitable for longer concept definitions and does not support the inspection of the occurring OWL elements via hyperlinks. OntoXpl's "NL description" seems to be more readable and carefully supports the inspection of mentioned entities via hyperlinks. Rice [13] offers visualization facilities for A-boxes where the complete graph structure of A-boxes is displayed. OntoXpl is restricted to a tree-like approximation due to the underlying SpaceTree tool [5] but it works better for larger A-boxes.

In our experience, OntoXpl's cross-referencing capabilities for hyperlinked concept, role, and individual names help users comprehend unknown ontologies faster than with the support offered by traditional editors.

## 5 Conclusion

In this paper we briefly introduced OntoXpl, a first step toward an OWL ontology exploration tool. OntoXpl is intended to complement ontology editors or other ontology visualization tools. A recently conducted informal experiment, where about 40 students had to design and implement 15 different OWL ontologies with a size of several hundred concept names, demonstrated that OntoXpl provides helpful information about ontologies that is otherwise not as easily available in ontology editors such as Protégé or OilEd. The implementation of the statistics command group is underway. It is also planned to integrate query results from nRQL such that individuals names are recognized as hyperlinks. Another important issue is the optimization of OntoXpl performance for larger ontologies containing thousands of concept names.

## References

[1] 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*. LNAI. Springer-Verlag, 2003.

[2] S. Bechhofer, I. Horrocks, and C. Goble. OilEd: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna*. LNAI Vol. 2174, Springer-Verlag, 2001.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[4] D. Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF Schema, http://www.w3.org/tr/2002/wd-rdf-schema-20020430/, 2002.

[5] J. Grosjean, C. Plaisant, and B. Bederson. SpaceTree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Procedings of IEEE Symposium on Information Visualization*, pages 57–64, Boston, USA, October 2002.

[6] V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, 2001.

[7] V. Haarslev and R. Möller. The Racer user's guide and reference manual, 2003.

[8] V. Haarslev and R. Möller. Optimization techniques for retrieving resources described in OWL/RDF documents: First results. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, June 2004.

[9] V. Haarslev, R. Möller, R. Van Der Straeten, and M. Wessel. Extended query facilities for Racer and an application to software-engineering problems. In *Proceedings of the International Workshop on Description Logics (DL-2004), Whistler, BC, Canada*, June 2004.

[10] O. Lassila and R.R. Swick. Resource description framework (RDF) model and syntax specification. recommendation, W3C, february 1999. http://www.w3.org/tr/1999/rec-rdf-syntax-19990222, 1999.

[11] T. Liebig and O. Noppens. OntoTrack: Fast browsing and easy editing of large ontologies. In *Proceedings of The Second International Workshop on Evaluation of Ontology-based Tools (EON2003), located at ISWC03*, Sanibel Island, USA, October 2003.

[12] Y. Lu. Exploration of OWL ontologies. Master's thesis, Department of Computer Science, Concordia University, Montreal, 2004 (in preparation).

[13] R. Möller, R. Cornet, and V. Haarslev. Graphical interfaces for Racer: querying DAML+OIL and RDF documents. In *Proc. International Workshop on Description Logics – DL'03*, 2003.

[14] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[15] D. Oberle, R. Volz, B. Motik, and S. Staab. An extensible ontology software environment. In *Handbook on Ontologies*, International Handbooks on Information Systems, chapter III, pages 311–333. Steffen Staab and Rudi Studer, Eds., Springer, 2004.

[16] Y. Sure, J. Angele, and S. Staab. OntoEdit: multifaceted inferencing for ontology engineering. *Journal on Data Semantics*, 2800/2003:128–152, 2003.

[17] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference, http://www.w3.org/tr/owl-guide/, 2003.

# Editing Description Logic Ontologies with the Protégé OWL Plugin

Holger Knublauch and Mark A. Musen
Stanford Medical Informatics, Stanford University, CA
holger@smi.stanford.edu, musen@smi.stanford.edu

Alan L. Rector
Medical Informatics Group, University of Manchester, UK
rector@cs.man.ac.uk

**Abstract**

The growing interest in the Semantic Web and the Web Ontology Language (OWL) will reveal the potential of Description Logics in industrial projects. The rich semantics of OWL provide powerful reasoning capabilities that help build, maintain and query domain models for many purposes. However, before OWL can unfold its full potential, user-friendly tools with a scalable architecture are required. We present the OWL Plugin, an extension of the Protégé ontology development environment, which can be used to define classes and properties, to edit logical class expressions, to invoke reasoners, and to link ontologies into the Semantic Web. We analyze some of the challenges for developers of Description Logic editors, and discuss some of our user interface design decisions.

## 1 Introduction

The formal underpinnings of Description Logics (DL) [1] are a cornerstone of Semantic Web technology such as the Web Ontology Language (OWL) [8]. DL reasoners can help build and maintain sharable ontologies by revealing inconsistencies, hidden dependencies, redundancies, and misclassifications [7].

While the formal terrain of DLs is now well mapped out and covered by efficient algorithms, a new generation of end-user tools is necessary to put this technology into the spotlight of industrial routine. In contrast to DLs, other modeling paradigms such as Object-Orientation and frames are supported by many professional editing tools that have evolved over many years of industrial use and, as a result, reflect best practices and common design patterns. A good

example of such a tool is Protégé [3], a knowledge modeling platform developed at Stanford Medical Informatics with support from a community of thousands of users over almost two decades.

Some of the technologies explored and refined by these tools can be applied to DL. For example, UML diagrams and Protégé's class explorer may serve as a good starting point for DL editors as well. However, there are crucial differences between DLs on the one hand and object-oriented or frame-based systems on the other that require custom-tailored solutions. In this paper, we will focus on four key issues in DL editing tools:

1. *Logical Expressions.* DL languages rely on (potentially deeply nested) logical expressions which can be hard to read, understand, and edit.

2. *Class Descriptions.* Classes in DL ontologies are either primitive or defined. A consistent approach for editing these two modes is needed.

3. *Reasoning.* DL ontologies can be classified (i.e., some relationships between concepts are inferred from the asserted class descriptions). A tool should provide both views, and allow users to compare the differences.

4. *Scalability.* DL ontologies are potentially very complex and large. An ontology editor should simplify navigation and help users to maintain large ontologies through mechanisms such as annotation metadata.

Some solutions for these issues have been explored by existing tools, most notably OilEd [2], developed at the University of Manchester. While OilEd succeeded in making DL technology available to a broader user community, its authors never intended it as a full ontology development environment but rather as a platform for experiments. As a result, OilEd's architecture is neither scalable to really large ontologies nor sufficiently flexible to allow its user interface to be customized.

The developers of Protégé and the OilEd team have recently joined forces in a transatlantic project called CO-ODE [1]. Our goal is to develop a new generation of OWL ontology editing tools, based on Protégé and the experiences collected with OilEd. Our collaboration has lead to the development of the Protégé OWL Plugin, which can be used to define classes and properties, to edit logical class expressions, to invoke reasoners, and to link ontologies into the Semantic Web.

The following sections provide details and design decisions of our current system. Due to space constraints we cannot provide a comprehensive description of the full system, but we focus on the key issues mentioned above. Before we go into these issues, we will start with some background on the Protégé architecture and its general principles.

[1] `http://www.co-ode.org`

# 2 Protégé and the OWL Plugin

Protégé [3] is an open-source ontology development environment with functionality for editing classes, slots (properties), and instances. The current version of Protégé (2.0) is highly extensible and customizable. At its core is a frame-based knowledge model [5] with support for metaclasses. Other languages such as OWL can be defined on top of this core frame model [6]. The mechanism we have used to represent the OWL metamodel in terms of Protégé frames will be described in a separate paper.

Protégé makes it not only possible to extend the metamodel but also to customize the user interface freely. As illustrated in Figure 1, Protégé's user interface consists of several screens, called *tabs*, each of which displays a different aspect of the ontology in a specialized view. Each of the tabs can include arbitrary Java components. Most of the existing tabs provide an explorer-style view of the model, with a tree on the left hand side and details of the selected node on the right hand side. The details of the selected object are typically displayed by means of forms. The forms consist of configurable components, called *widgets*. Typically, each widget displays one property of the selected object. There are standard widgets for the most common property types, but ontology developers are free to replace the default widgets with specialized components. Widgets, tabs, and back-ends are called *plugins*. Protégé's architecture makes it possible to add and activate plugins dynamically, so that the default system's appearance and behavior can be completely adapted to a project's needs.

The OWL Plugin[2] is a large Protégé plugin with support for OWL. It can be used to load and save OWL files in various formats, to edit OWL ontologies with custom-tailored graphical widgets, and to perform intelligent reasoning based on DLs. As shown in Figure 1, the OWL Plugin's user interface provides various default tabs. The *OWLClasses* tab displays the ontology's class hierarchy, allows developers to create and edit classes, and displays the result of the classification. The *Properties* tab can be used to create and edit the properties in the ontology. The *Individuals* tab can be used to create and edit individuals, and to acquire Semantic Web contents. The *Forms* tab allows to customize the forms used for editing classes, properties and individuals. The *Metadata* tab displays ontology metadata such as namespace prefixes. Ontology builders will typically focus on the OWLClasses tab, which is described in the following sections.

# 3 Editing Logical OWL Expressions

One of the first and most important decisions in the design of an OWL editor is how to display class expressions in a user-friendly but efficient way. The RDF

[2] `http://protege.stanford.edu/plugins/owl`

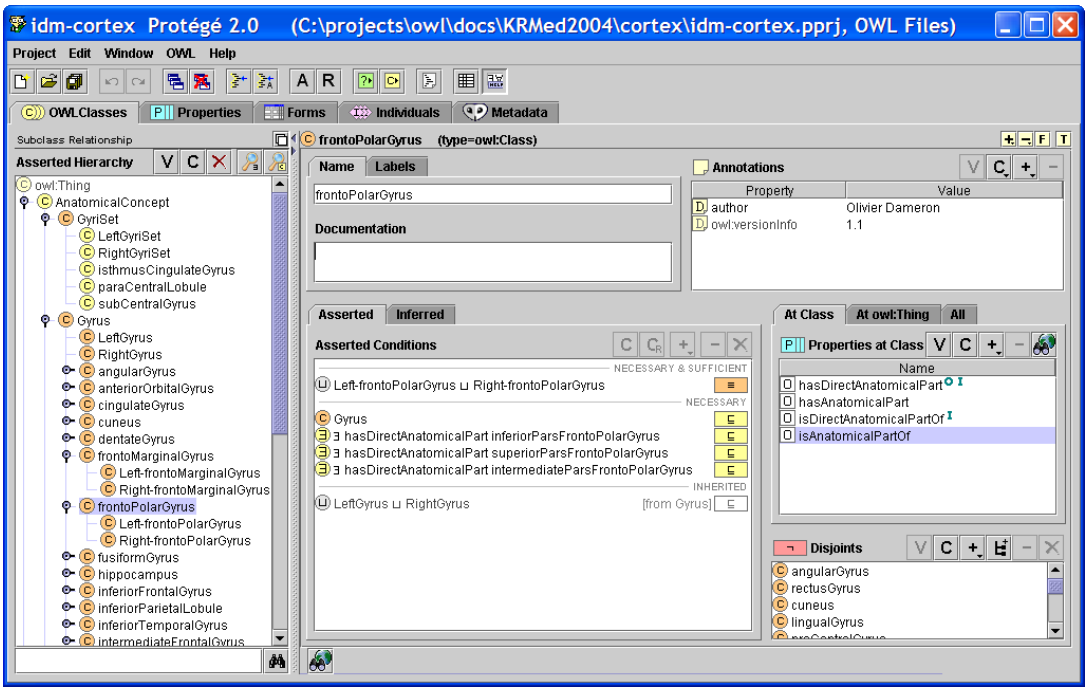



Figure 1: The class editor of the Protégé OWL Plugin.

syntax proposed in the OWL specification [8] is clearly too verbose to be of any use here. The OWL Abstract Syntax [9] is much more user-friendly, but still quite verbose. For the OWL Plugin, we chose to use an expression syntax based on standard DL symbols [1], such as ∀ and ⊔. These symbols (Figure 2) allow to the system display even complex nested expressions in a single row.

A trade-off from this syntax is that some characters are not found on standard keyboards. The OWL Plugin provides a comfortable expression editor which allows users to quickly assemble expressions with either the mouse or the keyboard (Figure 2). The special characters are mapped onto keys known from languages such as Java (e.g., `owl:intersectionOf` is entered with the `&` key). To simplify editing, keyboard users can exploit a syntax completion mechanism known from programming environments, which semi-automatically completes partial names after the uses has pressed `tab`. The expression editor is invoked by a double-click on a class expression, and then pops up directly below the expression. For really complex expressions, users can open a multi-line editor in an extra window, which formats the expression using indentation.

The OWL Plugin helps new users to get acquainted with the expression syntax. An English prose text is shown as a "tool tip" when the mouse is moved over the expression. For example, "∃ hasPet Cat" is displayed as "Any object which has a cat as its pet".

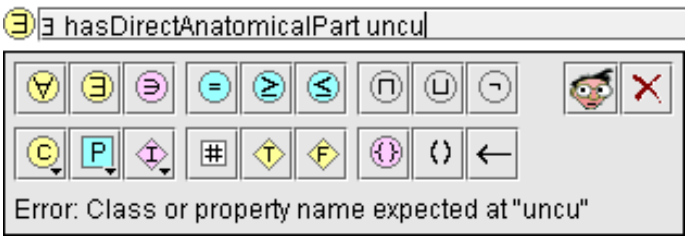



Figure 2: Protégé provides a comfortable editor for arbitrary OWL expressions.

# 4 Editing Class Descriptions

Another major design decision for a DL class editor is how to edit the logical class definitions. Protégé users are accustomed to an object-centered view to the interface which has required some effort to adapt to OWL. The distinction between defined and primitive classes simply is not found in frame-style or object-oriented modeling paradigms, and this can compound users' confusion when learning the DL paradigm. In the OWL specification, there is a lack of uniformity between defined classes (classes with necessary & sufficient conditions) and primitive classes (only necessary conditions). Multiple necessary conditions are represented by multiple `rdfs:subClassOf` statements whose intersection is implied, whereas sets of multiple necessary & sufficient conditions are represented by an `owl:equivalentClass` block containing an explicit intersection class. Although logically consistent, experience has shown that many users find the difference confusing. As a result, it was decided that the user interface should not simply reflect the structure suggested by the OWL specification but attempt to provide a clearer more uniform presentation to users.

During the evolution of the OWL Plugin we experimented with several interface designs, partly based on existing tools such as the Protégé core system and OilEd, partly on suggestions from our colleagues and users. OilEd has two modes: one to "partially" define a class with only necessary conditions, the other to "completely" define a class with necessary & sufficient conditions. There is a button to switch between these two modes. While this feature allowed the OilEd developers to provide customized widgets for various kinds of class descriptions (e.g. a widget for only restrictions), it has the disadvantage that users have to maintain separate class axioms in a separate pane for the necessary restrictions of classes that also are "completely" defined by a set of necessary & sufficient conditions. There is no one pane in OilEd in which one can see both the sets of necessary and sufficient conditions and any additional necessary conditions (i.e. axioms taking the defined class as their antecedent.)

As shown in the center of Figure 1, the OWL Plugin solves this problem by means of a list of conditions, organized into blocks of necessary & sufficient,

necessary, and inherited (i.e., inferred) conditions. Each of the necessary & sufficient blocks represents a single equivalent intersection class, and only those inherited conditions are listed that have not been further restricted higher up in the hierarchy (e.g., allValuesFrom Animal would not be shown if allValuesFrom Dog were also inferable).

The editor supports drag-and-drop between blocks in the conditions list, and copy-and-paste of expressions. It also supports changing superclasses by dragging a class from one parent to another in the class tree on the left hand side of the window.

In addition to the list of conditions, there is also a custom-tailored widget for entering disjoint classes, which has special support for typical design patterns such as making all siblings disjoint. This rather object-centered design of the OWLClasses tab makes it possible to maintain the whole class definition on a single screen.

# 5 Working with Classification

The OWL Plugin provides direct access to reasoners such as Racer [4]. The current user interface supports two types of DL reasoning: Consistency checking and classification (subsumption). Other types of reasoning, such as instance checking, are work in progress.

*Consistency checking* (i.e., the test whether a class could have instances) can be invoked either for all classes with a single mouse click, or for selected classes only. Inconsistent classes are marked with a red bordered icon.

*Classification* (i.e., inferring a new subsumption tree from the asserted definitions) can be invoked with the classify button on a one-shot basis. When the classify button is pressed, the system determines the OWL species, because some reasoners are unable to handle OWL Full ontologies. This is done using the validation service from the Jena[3] library. If the ontology is in OWL Full (e.g., because metaclasses are used) the system attempts to convert the ontology temporarily into OWL DL. The OWL Plugin supports editing some features of OWL Full (e.g., to assign ranges to annotation properties, and to create metaclasses). These are easily detected and can be removed before the data are sent to the classifier. Once the ontology has been converted into OWL DL, a full consistency check is performed, because inconsistent classes cannot be classified correctly. Finally, the classification results are stored until the next invocation of the classifier, and can be browsed separately. Classification can be invoked either for the whole ontology, or for selected subtrees only. In the latter case, the transitive closure of all accessible classes is sent to the classifier.

[3] `http://www.hpl.hp.com/semweb/jena2.htm`

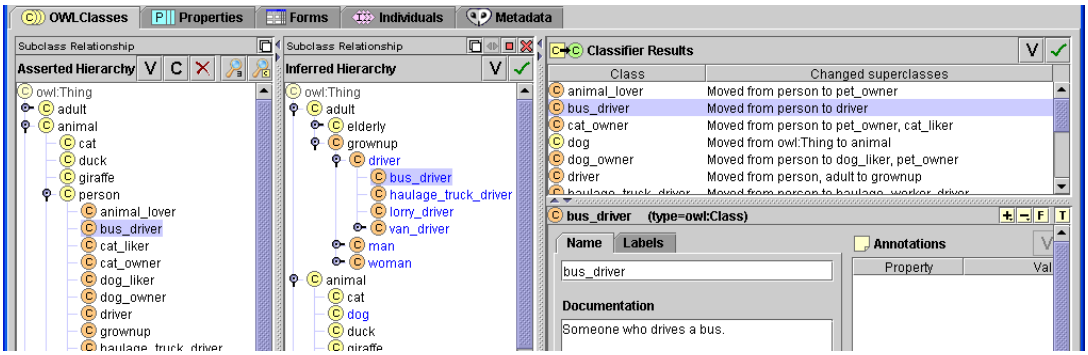



Figure 3: Protégé allows users to compare asserted and inferred class relationships. The system displays two hierarchies for asserted and inferred subsumption relationships, and provides a clickable list of the differences between them.

OWL files only store the subsumptions that have been asserted by the user. However, experience has shown that, in order to edit and correct their ontologies, users need to distinguish between what they have asserted and what the classifier has inferred. Many users may find it more natural to navigate the inferred hierarchy, because it displays the semantically correct position of the classes.

The OWL Plugin addresses this need by displaying both hierarchies and making available extensive information on the inferences made during classification. As illustrated in Figure 3, after classification the OWL Plugin displays an inferred classification hierarchy beside the original asserted hierarchy. The classes that have changed their superclasses are highlighted in blue, and moving the mouse over them explains the changes. Furthermore, a complete list of all changes suggested by the classifier is shown in the upper right area. A click on an entry navigates to the affected class. Also, the conditions widget can be switched between asserted and inferred conditions. All this allows the users to quickly analyze the changes.

# 6 Scalability

DL ontologies may become complex and large. The support for arbitrary class expressions means that DL ontologies typically contain many cross-links among classes, properties, individuals, and even among ontologies. This situation is complicated by the emerging Semantic Web, in which ontology development is distributed between groups around the world. As a result, scalability and support for ontology maintenance are crucial issues in ontology tools.

Protégé supports database storage that is scalable to several million concepts, and provides multi-user support for synchronous knowledge entry. The OWL

Plugin has already been used to edit ontologies with tens of thousands of classes. In support of handling such large ontologies, the OWL Plugin also provides a variety of navigation aids, such as lexical search functions and "find usage" buttons which allow to directly access all classes and properties that somehow reference a given object.

Documentation is essential for large ontologies. Most modeling or programming languages allow the attachment of comments or annotations to document the model's contents and to track provenance and changes. OWL supports this through annotation properties. The OWL Plugin allows to attach annotations to ontologies, properties, individuals, and classes, including anonymous classes. Annotation properties can be edited by means of a specific table widget (in the upper right area of Figure 1). The OWL Plugin allows the user to put arbitrary values into annotations, including complex objects. We are currently optimizing the tool for Dublin Core metadata so that, for example, annotation properties with change dates and authors will be filled in automatically.

# 7 Discussion

We have provided a brief overview of the OWL Plugin for Protégé. The Plugin explores several innovative approaches for displaying and editing logical expressions, editing class definitions, displaying classification results, and maintaining ontologies. Although we have not performed a formal evaluation of our user interface yet, the feedback from the user community is very encouraging. Many users suggest that the plugin's simple editors and comfortable reasoning interface supports rapid but sustainable ontology evolution. The constructive dialogs on forums such as those of the CO-ODE project and the Protégé discussion list will accelerate the evolution of the user interface with support for additional design patterns and best practices.

Many other groups from around the world are also developing Protégé plugins, including alternative wizard-style editors, and tools to query ontologies, to visualize ontologies graphically, and to manage ontology versions and changes. Furthermore, Protégé provides immediate access to all services for the Jena API, because it internally synchronizes the ontology with a Jena model.

Protégé has a large and rapidly growing community of thousands of users. Providing the community with a widely available, easy-to-use, open-source platform for OWL ontology design has the potential to use the leverage of that user base to bring DL technology into a wider audience. However, to do so effectively, requires thinking carefully about how to provide user interface metaphors with which that community is comfortable. The OWL Plugin aims to retain Protégé's object-centered interface without compromising OWL's DL semantics.

**Acknowledgements**. This work has been funded by a contract from the US National Cancer Institute and by grant P41LM007885 from the National Library of Medicine. Additional support for this work came from the UK Joint Information Services Committee under the CO-ODE grant. We would like thank our colleagues in Stanford (especially Ray Fergerson and Natasha Noy) and Manchester (especially Sean Bechhofer and Ian Horrocks for their work on OilEd and contribution to the interface discussions), the developers of the Jena library at HP Labs in Bristol, and the many Protégé users around the world for very valuable feedback during the evolution of the OWL Plugin.

# References

[1] F. Baader, D. Calvanese, D. McGuineness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[2] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able ontology editor for the Semantic Web. In *14th International Workshop on Description Logics*, Stanford, CA, 2001.

[3] J. Gennari, M. Musen, R. Fergerson, W. Grosso, M. Crubézy, H. Eriksson, N. Noy, and S. Tu. The evolution of Protégé-2000: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[4] V. Haarslev and R. Moeller. Racer: A core inference engine for the Semantic Web. In *2nd International Workshop on Evaluation of Ontology-based Tools (EON-2003)*, Sanibel Island, FL, 2003.

[5] N. Noy, R. Fergerson, and M. Musen. The knowledge model of Protégé-2000: Combining interoperability and flexibility. In *2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.

[6] N. Noy, M. Sintek, S. Decker, M. Crubézy, R. Fergerson, and M. Musen. Creating Semantic Web contents with Protégé-2000. *IEEE Intelligent Systems*, 2(16):60–71, 2001.

[7] A. Rector. Description logics in medical informatics. Chapter in [1].

[8] World Wide Web Consortium. OWL Web Ontology Language Reference. W3C Recommendation 10 Feb, 2004.

[9] World Wide Web Consortium. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 Feb, 2004.

# Reasoning Services for an OWL Authoring Tool: An Experience Report

Thorsten Liebig & Holger Pfeifer & Friedrich von Henke
Dept. of Artificial Intelligence
University of Ulm, Germany
{liebig|pfeifer|vhenke}@informatik.uni-ulm.de

## 1 Background

OWL has been designed to be a formal language for representing ontologies in the Semantic Web. In short, OWL is the result of combining an expressive Description Logic (DL) with techniques and standards of the Web. DLs have been well studied in the field of knowledge representation over the last decades. As one result, some highly optimized DL reasoners have been implemented, which provide an excellent starting point for building a sound and complete OWL DL/Lite reasoner. However, having a traditional DL system with standard functionality is not enough in the current context. So far, DL systems have been used by KR experts mainly in isolated application domains. Now, in order to make the Semantic Web happen far more flexible and interactive DL-based tools are needed for building, maintaining, linking, and applying ontologies even for non-experienced users. The importance of so-called non-standard inference services that support building and maintaining knowledge bases has been pointed out recently [1, 2]. We argue that the availability of those inference services is a fundamental premise for upcoming real-world Semantic Web systems and applications. Our experience in the course of developing the graphical ontology editor OntoTrack is a prime example here.

## 2 OntoTrack: a Novel Ontology Editor

OntoTrack [7] is a new browsing and editing "in-one-view" ontology authoring tool for OWL Lite that combines a sophisticated graphical layout with mouse-enabled editing features and instant reasoning feedback using an external DL reasoner. More precisely, all user changes after each editing step are send to the RACER [6] reasoner via a TCP-based client interface. The reasoner will then make all modeling consequences explicitly available. OntoTrack will hand over relevant consequences (e.g. new subsumption relationships, equivalent or unsatisfiable classes) to the user by providing appropriate graphical feedback. However, implementing this feedback functionality turned out to become difficult and even impossible for some language statements (e.g. the deletion of global domain and range restrictions of properties couldn't be implemented due to a missing retraction functionality).

Currently, DL reasoners only provide some kind of batch-oriented enter and query interface. Because of lack of algorithms for appropriately handling incremental additions to a knowledge base [9] complete reclassification after each user interaction is necessary. Furthermore, in order to become aware of a new subsumption relationship due to a just added property restriction for example, OntoTrack needs to query the reasoner about direct super classes for almost all classes of the ontology in turn. One could of course narrow this set to those classes that also have an explicit or inherited restriction on that particular property or a sub-property thereof. But this requires to have explicit knowledge about inherited restrictions or sub-properties, which in turn may result in additional queries. Deletion of, or changes within, classes and properties or even fractions thereof is an analogous problem. However, using an optimized tableaux-style reasoner for a language with an expressivity comparable to that of OWL, retraction and changing of definitions (e. g. GCIs) may be of high complexity because of optimization techniques like absorption.

## 3 Desirable Reasoning Services

Based on our experiences in the course of developing OntoTrack, we briefly summarize our application requirements with respect to DL reasoners for supporting ontology editing.

Instead of querying for all possible changes with respect to a specific consequence (most notably the direct subsumption relationship) after each editing step we would like to have an event-triggered notification model on the reasoner side. This mechanism should only publish the set of differences in conclusions with respect to the previous state. This would correspond to a likewise TBox technique of RACERs ABox publish-subscribe mechanism.

Another desirable feature is incremental reasoning and retraction of definitions. As long as partial class definitions are concerned, additive incremental reasoning can be done with help of additional GCIs and reclassification. However, adding a global domain or range restriction will then result in GCIs which are not absorbable. Incremental reasoning with complete class definitions requires to retract the original definition before adding a new restriction in combination with the original one. As mentioned before, retraction of definitions or statements may be of high cost but is a prerequisite for interactive ontology tools. A solution could consist of a reasoner heuristic that analyzes the retraction statement and decides about on-the-fly deletion or reclassification. A related problem is how to detect and deal with statements explicitly or implicitly affected by a retraction process e. g. due to references.

A serious issue of each ontology authoring tool is concerned with debugging of ontologies. Here standard inference services provide no help to resolve inconsistencies in logical incoherent ontologies. In [12] a new reasoning service for pinpointing logical contradictions of $\mathcal{ALC}$ ontologies has been developed.

Methods for explaining unsatisfiability of classes and class subsumption have also been developed for $\mathcal{ALC}$ [11, 5]. Sophisticated debugging or explanation services in combination with an appropriate graphical user interface would obviously make

ontology authoring much more efficient. An on-demand generation of an ABox model for a selected class may also be helpful for explanation.

Other novel inference services intended to support building an ontology have been developed (see sec. 6.3 in [3] for a summary). One interesting service consists of matching of class patterns against class descriptions in order to find already defined classes with a similar structure. Another approach tries to create class definitions by generalizing one or more user given ABox assertions. Other non-standard inference services like least common subsumer or most specific concept are also relevant during authoring of ontologies.

Unfortunately, only some of these non-standard reasoning services have been implemented[1] and only a few are found in state of the art reasoning systems today. First this is due to the fact that some of them only make sense if used for DLs less expressive than OWL. Second, approaches for solving these services are usually based on structural subsumption algorithms known to be not appropriate for languages like OWL. However, only some ontologies use all available language constructs. A large fraction is within restricted clusters of less expressive sublanguages [13]. We therefore hope to see some of these reasoning features (even for sublanguages of OWL) integrated into reasoners in the future.

Technical requirements with respect to interfaces and communication are also an important issue for building a successful application. A state of the art architecture should support multiple clients via standard protocols. Notably, some of the most recent reasoner developments are either not network-aware or without any interface documentation. Support of standard formats (e. g. KRSS [10], DIG [4]) and native OWL import either from file or via HTTP is also a desired quality.

## 4 Conclusion

The development of sophisticated and adequate Semantic Web tools for end users strongly depends on sufficiently broad reasoning services and appropriate interfaces of its core technology, namely DL reasoners. It is worth mentioning that our experiences are not specific to our choice for RACER as external DL reasoner. An internal evaluation of some DL based reasoning systems potentially capable of handling portions of OWL (FaCT, FaCT++, RACER, Pellet, BOR) identified general deficits with respect to our requirements mentioned above. We therefore argue that not only correctness, efficiency, and language conformity are important, service diversity and interactive capabilites should become an issue for research and criteria of future reasoner evaluations.

## References

[1] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics as Ontology Languages for the Semantic Web. In *Festschrift in honor of Jörg Siekmann*. Springer, 2003.

[1] The Classic system is a notable exception here [8].

[2] Franz Baader, Ian Horrocks, and Ulrike Sattler. *Handbook on Ontologies*, chapter Description Logics, pages 3–28. International Handbooks on Information Systems. Springer Verlag, 2004.

[3] Franz Baader, Ralf Küsters, and Frank Wolter. *The Description Logic Handbook*, chapter Extensions to Description Logics. Cambridge University Press, 2003.

[4] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG Description Logic Interface. In *Proc. of International Workshop on Description Logics (DL2003)*, Rome, Italy, 2003.

[5] Alex Borgida, Enrico Franconi, and Ian Horrocks. Explaining $\mathcal{ALC}$ subsumption. In *Proc. of International Workshop on Description Logics (DL1999)*, Linköping, Sweden, 1999.

[6] Volker Haarslev and Ralf Möller. RACER System Description. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001)*, volume 2083 of *LNAI*, pages 701–706, Siena, Italy, 2001.

[7] Thorsten Liebig and Olaf Noppens. OntoTrack: Fast Browsing and Easy Editing of Large Ontologies. In *Proc. of The 2nd Int. Workshop on Evaluation of Ontology-based Tools (EON2003)*, Sanibel Island, USA, 2003.

[8] Deborah L. McGuinness and Alex Borgida. Explaining Subsumption in Description Logics. Technical Report LCSR-TR-228, Dept. of Computer Sciences, Rutgers University, September 1994.

[9] Ralf Möller and Volker Haarslev. *The Description Logic Handbook*, chapter Description Logics Systems. Cambridge University Press, 2003.

[10] Peter F. Patel-Schneider and Bill Swartout. Description Logic Specification from the KRSS Effort. Working version (draft), 1993.

[11] Stefan Schlobach and Ronald Cornet. Explanation of Terminological Reasoning A Preliminary Report. In *Proc. of International Workshop on Description Logics (DL2003)*, Rome, Italy, 2003.

[12] Stefan Schlobach and Ronald Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In *Proc. of the Belgian-Dutch Conference on AI (BNAI03)*, Nijmegen, The Netherlands, 2003.

[13] Christoph Tempich and Raphael Volz. Towards a benchmark for Semantic Web reasoners - an analysis of the DAML ontology library. In *Proc. of The 2nd Int. Workshop on Evaluation of Ontology-based Tools (EON2003)*, Sanibel Island, USA, 2003.

# Towards Implementing Finite Model Reasoning in Description Logics

Marco Cadoli[1], Diego Calvanese[2], Giuseppe De Giacomo[1]
[1] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
*lastname*@dis.uniroma1.it

[2] Faculty of Computer Science
Free University of Bolzano/Bozen
Piazza Domenicani 3, I-39100 Bolzano, Italy
calvanese@inf.unibz.it

**Abstract**

We describe our ongoing work that aims at understanding how one can develop a system that performs finite model reasoning in Description Logics. In particular we report on the preliminary results that we have obtained by encoding reasoning services in DLs as specifications for constraint programming solvers. We have used such an approach to reason with respect to finite models on UML class diagrams.

## 1 Introduction

Expressive Description Logics (DLs) do not enjoy the finite model property. This means that a knowledge base expressed in such a DL may be satisfiable, though it admits only models with an infinite domain [9, 1]. The loss of the finite model property is due to the interaction between cardinality constraints, the use of direct and inverse roles, and general (possibly cyclic) inclusion assertions in the knowledge base. Hence, for such DLs, techniques have been investigated to address reasoning with respect to finite models only [10, 6, 8, 13]. Notably, the presence of cardinality constraints, together with the requirement of models to be finite, gives rise to combinatorial aspects in the reasoning problems, and such aspects are taken into account by the proposed techniques by resorting to the encoding of satisfiability into solving numerical dependencies.

The recent interest in DLs as a means to formalize and reason on class based formalisms for conceptual modeling, software engineering, and ontologies, has revived the interest in finite model reasoning, since such formalisms are often used to represent structures that are intrinsically finite (e.g., a databases, object repositories, etc.).

In this paper we focus on UML class diagrams.[1] Finite model reasoning in UML

[1] http://www.omg.org/uml/

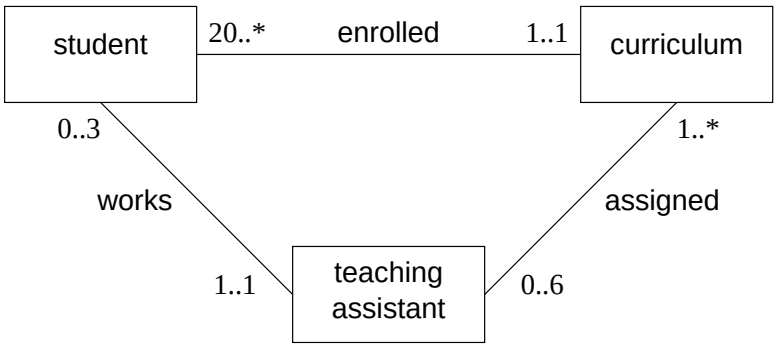



Figure 1: A finitely inconsistent class diagram

class diagrams is of crucial importance for assessing quality of the analysis phase in software development. Finite unsatisfiability of a class diagram means that there are some classes or associations which cannot have a finite number of instances. This concept must be contrasted with unrestricted unsatisfiability, which means that classes or associations cannot have *any*, i.e., both finite and infinite, number of instances. Indeed, for the purpose of software engineering, finite model reasoning in UML class diagrams is often considered more important than unrestricted reasoning.

**Example 1** As an example, consider Figure 1, which states reasonable constraints for a university scenario; e.g., there is a binary association *enrolled* between classes *student* and *curriculum*, and the *multiplicities* on the association express that each student is enrolled in exactly one curriculum, while each curriculum must enroll at least 20 students ('$*$' denotes that there is no constraints on the maximum multiplicity). Such a diagram is satisfiable in the unrestricted sense. Indeed, suppose there is an instance $c_1$ of *curriculum*; then there must be at least 20 students enrolled in $c_1$, and since each such student must work with exactly one teaching assistant and each teaching assistant can work with at most 3 students, there must be at least 7 teaching assistants; since at most 6 teaching assistants can be assigned to $c_1$, there must be a second instance $c_2$ of *curriculum*; considering that each student can be enrolled in at most one curriculum, by applying the same line of reasoning, we can see that there must be an infinite sequence of instances $c_3, c_4, \ldots$ of *curriculum*. However, this shows also that the diagram in Figure 1 is *not* finitely satisfiable. ∎

In this paper we address the implementation of finite model reasoning in DLs, a task that has not been attempted so far. As a matter of fact, state-of-the-art DL reasoning systems, such as FACT or RACER, perform unrestricted reasoning, and do not address finite model reasoning. Interestingly, finite model reasoning on UML class diagrams itself has, to the best of our knowledge, never been implemented in any kind of system.

The main result of this paper is that it is possible to use off-the-shelf tools for constraint modeling and programming for obtaining a finite model reasoner. In particular, exploiting the finite model reasoning technique presented in [10, 8], we propose

an encoding as Constraint Satisfaction Problem (CSPs) of knowledge base satisfiability. Moreover, we show also how CSP can be used to actually return a finite model of the knowledge base (a task needed for particular applications, as [4]).

In fact, in this paper, we focus on UML class diagrams without ISA relations between associations, seen as a means of describing DL knowledge bases (see, e.g., [3]). More precisely, such UML class diagrams correspond essentially to *primitive* $\mathcal{ALUNI}$ *knowledge bases*, which consist of (possibly cyclic) inclusion assertions of the form $B \sqsubseteq C$, where $B$ is boolean combination of atomic concepts and $C$ is a concept of the DL $\mathcal{ALUNI}$ [10]. Experimentation so far is in a preliminary stage, but the results we have obtained are quite encouraging.

## 2 Finite Model Reasoning in DLs

The technique for finite model reasoning in DLs with number restrictions, inverse roles and inclusion assertions was first presented in [10, 6], and is based on translating the knowledge base in a set of linear inequalities. Intuitively, consider a simple knowledge base $\mathcal{K}$ formed by inclusion assertions

$$\begin{array}{lcl} \top & \sqsubseteq & \forall R^{-}.A \sqcap \forall R.B \\ A & \sqsubseteq & (\geq m_1\, R) \sqcap (\leq n_1\, R) \\ B & \sqsubseteq & (\geq m_2\, R^{-}) \sqcap (\leq n_2\, R^{-}) \end{array}$$

for each role $R$. Such assertions express that $R$ is typed on $A$ for the first component and $B$ for the second, and additionally expresses minimum and maximum cardinality constraints on the participation to $R$. It is easy to see that such a knowledge base $\mathcal{K}$ is always satisfiable (assuming $m_i \leq n_i$) if we admit infinite models. Hence, only finite model reasoning is of interest. We observe that, if $\mathcal{K}$ is finitely satisfiable, then it admits a finite model in which all atomic concepts are pairwise disjoint. Exploiting this property, we can encode finite satisfiability of $\mathcal{K}$ in a constraint system as follows. We introduce one variable for each role and atomic concept, representing the number of instances of the role (resp., concept) in a possible model of $\mathcal{K}$. Then, for each $R$ we introduce the constraints

$$\begin{array}{rcl} m_1 \cdot a & \leqslant & r \leqslant n_1 \cdot a \\ m_2 \cdot b & \leqslant & r \leqslant n_2 \cdot b \\ a \cdot b & \geqslant & r \end{array}$$

where $a$, $b$, and $r$ are the variables corresponding to $A$, $B$, and $R$, respectively.

It is possible to show that, from a solution of such a constraint system, we can construct a finite model of $\mathcal{K}$ in which the cardinality of the extension of each concept and role is equal to the value assigned to the corresponding variable[2].

The approach above can be extended to deal also with inclusion assertions expressing ISA, disjointness, and covering between concepts. Intuitively, one needs to introduce one variable for each combination of atomic concepts; similarly, for roles

[2]In fact, if one is interested just in the existence of a finite model, one could drop the nonlinear constraints of the form $a \cdot b \geqslant r$.

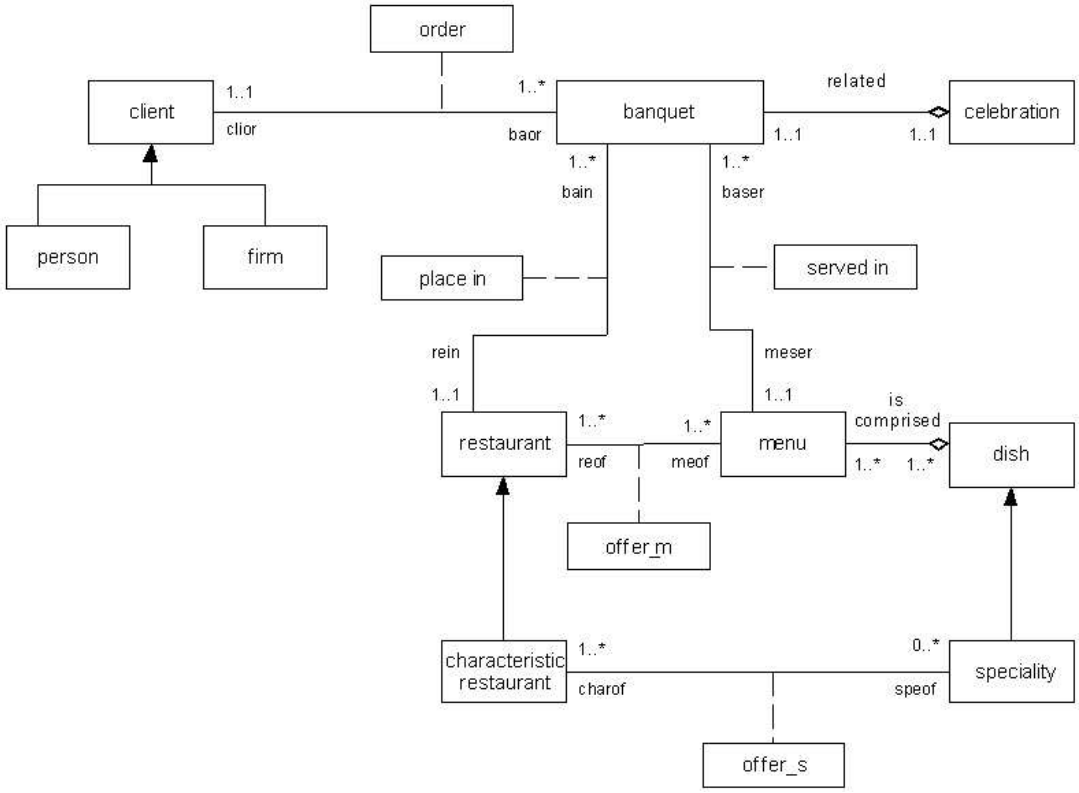



Figure 2: The "restaurant" UML class diagram

one needs to distinguish how, among the possible combinations of atomic concepts, the role is typed in its first and its second component. This leads, in general, to the introduction of an exponential number of variables and constraints [10]. We illustrate this, in the context of UML, in the next section.

## 3 Finite Model Reasoning via CSP

We address the problem of finite satisfiability of UML class diagrams, and show how it is possible to encode two problems as constraint satisfaction problems (CSPs), namely:

1. deciding whether all classes in the diagram are simultaneously finitely satisfiable, and

2. finding –if possible– a finite model with non-empty classes and associations.

We use the "restaurant" class diagram, shown in Figure 2, as our running example.

First we address the problem of deciding finite satisfiability. As mentioned, we use the technique proposed in [6], which is based on the idea of translating the multiplicity constraints of the UML class diagram into a set of inequalities among integer variables.

The variables and the inequalities of the CSP are modularly described considering in turn each association of the class diagram. Let $A$ be an association between classes $C_1$ and $C_2$ such that the following multiplicity constraints are stated:

- there are at least $min_1$ and at most $max_1$ links of type $A$ (instances of the association $A$) for each object of the class $C_1$;
- there are at least $min_2$ and at most $max_2$ links of type $A$ for each object of the class $C_2$.

Referring to Figure 2, if $A$ stands for *served_in*, $C_1$ stands for *banquet*, and $C_2$ stands for *menu*, then $min_1$ is 1, $max_1$ is 1, $min_2$ is 1, and $max_2$ is $\infty$.

For the sake of simplicity, we start from the special case in which neither $C_1$ nor $C_2$ participates in a ISA hierarchy, e.g., the *related* and the *served_in* associations of Figure 2.

The CSP is defined as follows:

- there are three non-negative variables `c1`, `c2`, and `a`, which stand for the number of objects of the classes $C_1$ and $C_2$ and the number of links of $A$, respectively;
- there are the following constraints (we use the syntax of the constraint programming language OPL [14]):

```
1. min_1 * c1 <= a;
2. max_1 * c1 >= a;
3. min_2 * c2 <= a;
4. max_2 * c2 >= a;
5. a  <= c1 * c2;
6. a  >= 1;
7. c1 >= 1;
8. c2 >= 1;
```

Constraints 1–4 account for the multiplicity of the association; they can be omitted if either `min_`$i$ $= 0$, or `max_`$i$ $= \infty$ (symbol '`*`' in the class diagram). Constraint 5 sets an upper bound for the number of links of type $A$ with respect to the number of objects. Constraints 6–8 define the satisfiability problem we are interested in: we want at least one object for each class and at least one link for each association. The latter constraints can be omitted by declaring the variables as strictly positive.

When either $C_1$ or $C_2$ are involved in ISA hierarchies, the constraints are more complicated, because the meaning of the multiplicity constraints changes. As an example, the multiplicity `1..*` of the *order* association in Figure 2 states that a *client* order*s* at least one *banquet*, but the client can be a *person*, a *firm*, both, or neither (assuming the generalization is neither disjoint nor complete). In general, for an ISA hierarchy involving $n$ classes, $O(2^n)$ non-negative variables corresponding to all possible combinations must be considered. For the same reason, we must consider four distinct specializations of the *order* association, i.e., one for each possible combination. Summing up, we have the following non-negative variables:

- `person`, `order_p`, for clients who are persons and not firms;
- `firm`, `order_f`, for clients who are firms and not persons;
- `person_firm`, `order_pf`, for clients who are both firms and persons;
- `client`, `order_c`, for clients who are neither firms nor persons;

plus the positive `banquet` variable.

The constraints (in the OPL syntax) which account for the *order* association are as follows:

```
/*  1 */  client <= order_c;
/*  2 */  firm <= order_f;
/*  3 */  person <= order_p;
/*  4 */  person_firm <= order_pf;
/*  5 */  banquet = order_c + order_f + order_p + order_pf;
/*  6 */  order_c <= client * banquet;
/*  7 */  order_f <= firm * banquet;
/*  8 */  order_p <= person * banquet;
/*  9 */  order_pf <= person_firm * banquet;
/* 10 */  client + firm + person + person_firm >= 1;
/* 11 */  order_c + order_f + order_p + order_pf >= 1;
```

Constraints 1–4 account for the '1' in the `1..*` multiplicity; Constraint 5 translates the `1..1` multiplicity; Constraints 6–9 set an upper bound for the number of links of type *order* with respect to the number of objects; Constraints 10–11 define the satisfiability problem (`banquet` is already strictly positive).

We refer the reader to [8, 6] for formal details of the translation, and in particular for the proof of its correctness. As for the implementation, the "restaurant" example has been encoded in OPL as a CSP with 24 variables and 40 constraints. The solution has been found by the underlying constraint programming solver, i.e., ILOG's SOLVER [12, 11], in less than 0.01 seconds.

## 4 Constructing a Finite Model

We now turn to the second problem, i.e., finding –if possible– a finite model with non-empty classes and associations. The basic idea is to encode in the constraint modeling language of OPL the semantics of the UML class diagram (see [3, 2]). In particular we use arrays of boolean variables representing the extensions of predicates, where the size of the arrays is determined by the output of the first problem. Since in the first problem we have enforced the multiplicity constraints, and obtained an admissible number of objects for each class (actually for each combination of classes), we know that a finite model of the class diagram exists. We also know the size of the universe of such a finite model, which is equal to the sum, over all combinations of classes, of the number of objects in each combination of classes (recall that each combination of classes is disjoint from all other combinations of classes).

Referring to our "restaurant" example, we have the following declarations describing the size of our universe and two sorts:

```
int size = client + person + firm + person_firm + restaurant + menu +
```

```
 characteristic_restaurant + dish + specialty + banquet + celebration;
range Bool 0..1;
range Universe 1..size;
```

The arrays corresponding, e.g., to the *client* and *banquet* classes, and to the *order_c* association are declared as follows:

```
var Bool Client[Universe];
var Bool Banquet[Universe];
var Bool Order_C[Universe,Universe];
```

Now, we have to enforce some constraints to reflect the semantics of the UML class diagram [3, 2], namely that:

1. each object belongs to exactly one class;
2. the number of objects (resp., links) in each class (resp., association) is coherent with the solution of the first problem;
3. the associations are *typed*, e.g., that a link of type *order_c* insists on an object which is a *banquet* and on another object which is a *client*;
4. the multiplicity constraints are satisfied.

Such constraints can be encoded as follows (for brevity, we show only some of the constraints).

```
// AN OBJECT BELONGS TO ONE CLASS
forall(x in Universe)
  Client[x] + Person[x] + Firm[x] + Person_Firm[x] + Restaurant[x] +
  Characteristic_Restaurant[x] + Dish[x] + Specialty[x] + Banquet[x] +
  Celebration[x] + Menu[x] = 1;
// ENFORCING SIZES OF CLASSES AND ASSOCIATIONS
sum(x in Universe) Client[x] = client;
sum(x in Universe) Banquet[x] = banquet;
sum(x in Universe, y in Universe) Order_C[x,y] = order_c;
// TYPES FOR ASSOCIATIONS
forall(x, y in Universe)
  Order_C[x,y] => Client[x] & Banquet[y];
// MULTIPLICITY CONSTRAINTS ARE SATISFIED
forall(x in Universe)
  Client[x] => sum(y in Universe) Order_C[x,y] >= 1;
```

Summing up, the "restaurant" example has been encoded in OPL with about 40 lines of code. After instantiation, this resulted in a CSP with 498 variables and 461 constraints. The solution has been found by ILOG's SOLVER in less than 0.01 seconds, and no backtracking.

## 5 Notes on Complexity

Few notes about the computational complexity are in order. It is known that solving both problems of deciding finite satisfiability and finite model finding for primitive $\mathcal{ALUNI}$ knowledge bases (and hence for UML class diagrams) are EXPTIME-complete [10, 7, 8]. Our encoding of the first problem in a CSP may result in a

number of variables which is exponential in the size of the diagram. Anyway, since the exponentiality depends on the maximum number of classes involved in the same ISA hierarchy, the actual size for real UML diagrams will not be very large.

As for the second problem, our encoding is polynomial in the size of the class diagram. Note that this does not contradict the EXPTIME lower bound, due to the *program complexity* of modeling languages such as OPL. Indeed, in [5] it is shown that the program complexity of boolean linear programming is NEXPTIME-hard.

## 6 Conclusions

In this paper we have reported on our ongoing investigation on implementing finite model reasoning in DLs. We have shown how current state-of-the-art constraint solvers can be used to perform such a kind of reasoning. The performance of such systems in the experiments done so far is quite good, though these results still need to be confirmed on larger cases, such as the CIM[3] diagrams. This is ongoing work. We are also building a prototype system for finite model reasoning that uses OPL as reasoning engine.

Our implementation can so far not deal with UML class diagrams containing associations between roles. The translation of such diagrams in a DL knowledge base requires to introduce inclusion assertions between roles, or, alternatively, to reify roles and thus introduce qualified number restrictions to encode multiplicities [2]. In [8, 6], an extension of the technique for finite model reasoning illustrated here is proposed, that can deal also with qualified number restrictions. However, such a method requires to introduce a number of variables and constraints that is double exponential in the size of the knowledge base. In [13] a more involved technique is presented, for which the size of the constraint system stays simply exponential. However, numbers appearing in number restrictions cannot be dealt with directly in the constraints and need to be encoded using counters. Thus, the possibility of actually implementing one or the other of these methods by making use of constraint solvers requires further investigation.

## References


[1] 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.

[2] D. Berardi, A. Calì, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. Technical Report 11-03, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2003.

[3] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams is EXPTIME-hard. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*,


[3] `http://www.dmtf.org/standards/cim/`

pages 28–37. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-81/, 2003.

[4] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. *e*-Service composition by description logics based reasoning. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, pages 75–84. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-81/, 2003.

[5] M. Cadoli. The expressive power of binary linear programming. In *Proc. of the 7th Int. Conf. on Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *Lecture Notes in Computer Science*, 2001.

[6] D. Calvanese. Finite model reasoning in description logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 292–303, 1996.

[7] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proc. of the 12th Eur. Conf. on Artificial Intelligence (ECAI'96)*, pages 303–307. John Wiley & Sons, 1996.

[8] D. Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1996. Available at http://www.dis.uniroma1.it/pub/calvanes/thesis.ps.gz.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 23, pages 1581–1634. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

[10] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 109–120, 1994.

[11] ILOG Solver system version 5.1 user's manual, 2001.

[12] ILOG OPL Studio system version 3.6.1 user's manual, 2002.

[13] C. Lutz, U. Sattler, and L. Tendera. The complexity of finite model reasoning in description logics. In *Proc. of the 19th Int. Conf. on Automated Deduction (CADE 2003)*, pages 60–74, 2003.

[14] P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.

# *DL-Lite*: Practical Reasoning for Rich DLs

Diego Calvanese[1], Giuseppe De Giacomo[2], Maurizio Lenzerini[2],
Riccardo Rosati[2], Guido Vetere[3]

[1] Faculty of Computer Science
Free University of Bolzano/Bozen
Piazza Domenicani 3
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

[2] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
I-00198 Roma, Italy
*lastname*@dis.uniroma1.it

[3] IBM Italia
Via Sciangai 53, I-00144 Roma, Italy
guido_vetere@it.ibm.com

**Abstract**

In this paper we study a DL rich enough to express UML class diagrams including ISA and disjointness between classes (but not covering constraints), typing of associations, and participation and functional cardinality constraints. For such a DL, which we call *DL-Lite*, we propose novel reasoning techniques for a variety of tasks, notably including query containment and query answering for conjunctive queries over concepts and roles. The techniques are based on query containment under constraints typical of databases. A prototype implementation of *DL-Lite* has been developed and experimented with.

## 1 Introduction

One of the most important lines of research in Description Logics (DLs) is concerned with the trade-off between expressive power and computational complexity of sound and complete reasoning. Research on this topic has shown that DLs with efficient, i.e., worst-case polynomial time, reasoning algorithms lack modeling power required in capturing conceptual models and basic ontology languages, while DLs with sufficient modeling power suffers from inherently worst-case exponential time behavior of reasoning [8, 9, 2].

In this paper we propose a new DL specifically tailored to capture conceptual data models (e.g., Entity-Relationship) [1], Object-oriented formalisms (e.g., basic UML class diagrams)[1], and basic ontology languages. Notably, we show that advanced forms of sound and complete reasoning, taking into account a knowledge base constituted by a TBox and an ABox, and queries, can be done in polynomial time in the size of the knowledge base. More precisely, our contributions are the following:

1. We define *DL-Lite*, a DL rich enough to capture a significant ontology language. In particular, *DL-Lite* is able to express UML class diagrams including ISA

[1] http://www.omg.org/uml/

and disjointness between classes (but not covering constraints), typing of associations, and cardinality constraints imposing mandatory participation to roles and functionality of roles.

2. For such a DL we propose novel reasoning techniques for a variety of tasks, including conjunctive query answering and containment between conjunctive queries over concepts and roles. Our presentation is focused on the problem of answering conjunctive queries over a knowledge base. We observe that this is one of the few results on answering complex queries (i.e., not corresponding simply to a concept or a role) over a knowledge base [6, 7]. Indeed, answering conjunctive queries over a knowledge base is a challenging problem, even in the case of DL-lite, where the combination of constructs expressible in the knowledge base does not pose particular difficulties in computing subsumption. Our solution bu builds upon and extends a series of techniques developed in databases for query containment and query answering under constraints [10, 3, 4].

3. We show that the above mentioned reasoning tasks can be carried out in polynomial time in the size of the knowledge base.

A prototype implementation of *DL-Lite* has been developed and tested within the SMO (System Management Ontology) project carried out jointly by the University of Rome "La Sapienza" and the IBM Tivoli Laboratory.

The next section defines *DL-Lite* and the associated reasoning services. Section 3 shows that *DL-Lite* is indeed an interesting logic for capturing the basic modeling constructs of conceptual models and ontology languages. Section 4 briefly describes the fundamental reasoning technique associated to *DL-Lite*. Section 5 concludes the paper.

## 2 *DL-Lite*

The DL *DL-Lite* that we present in this paper is quite simple from the language point of view. Namely, starting from atomic concepts, denoted by $A$, possibly with subscripts, and atomic roles, denoted by $R$, possibly with subscripts, we define *basic concepts*, denoted by $B$, as follows:

$$B \ ::= \ A \ \mid \ \exists R \ \mid \ \exists R^{-}$$

where $A$ is an atomic concept, $\exists R$ is the usual unqualified existentiality on atomic roles $R$, and $\exists R^{-}$ is the same on *inverse roles*. General concepts in DL-lite are then defined as follows:

$$C \ ::= \ B \ \mid \ \neg B \ \mid \ C_1 \sqcap C_2$$

Note that we have negation on basic concept only and that we have conjunction but not disjunction.

Using this simple language we allow to make assertions of specific forms. Specifically, in a *DL-Lite* TBox, we allow for *inclusion assertions* of the form:

$$B \sqsubseteq C$$

where on the left-hand-side we must have a basic concept ($B$), while on the right-hand-side we may have a general *DL-Lite* concept.

Observe that we do allow for cyclic assertions. Indeed, we can enforce the cyclic propagation of the existence of an $R$-successor using the two *DL-Lite* inclusion assertions $A \sqsubseteq \exists R$, $\exists R^- \sqsubseteq A$. The constraint imposed on a model is similar to the one imposed by the $\mathcal{ALC}$ cyclic assertion $A \sqsubseteq \exists R.\top \sqcap \forall R.A$, though stronger, since it additionally enforces the second component of $R$ to be typed by $A$.

Also, in addition to inclusion assertions, in *DL-Lite* we have *functionality assertions* of the form

$$(\mathsf{funct}\ R), \qquad (\mathsf{funct}\ R^-)$$

expressing, respectively, the functionality of atomic roles and of inverses of atomic roles.

As for the ABox, we allow for membership assertions on atomic concept and on atomic roles:

$$A(a), \qquad R(a, b)$$

stating respectively that the object (denoted by the constant) $a$ is and instance of $A$ and that the pair $(a, b)$ of objects (denoted by the two constants $a$ and $b$) is an instance of the atomic role $R$.

In fact, to denote objects, *DL-Lite* includes two kinds of constants: the usual constants for which the unique name assumption holds, and the so called *soft constants*, which are constants for which the unique name assumption does not hold.

Notice that, using soft constants, we can express in the ABox also membership assertions involving existentials. For example, to express the membership assertion $(\exists R)(a)$, where $a$ is a non-soft constant, we can include in the ABox the membership assertion $R(a, u)$ where $u$ is a fresh (i.e., not used elsewhere in the ABox) soft constant.

Given a *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, we can query the knowledge base using conjunctive queries. A conjunctive query $q$ is an expression of the form

$$\{\, \vec{x} \mid conj(\vec{x}, \vec{y}) \,\}$$

where $\vec{x}$ are the so called distinguished variables that will be bound with object in the KB, $\vec{y}$ are the non-distinguished variables, which are existentially qualified variables, and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $A(z)$ or $R(z_1, z_2)$ where $A$ and $R$ are respectively atomic concept and roles and $z$, $z_1$, $z_2$ are either constants in the KB or variables in $\vec{x}$ or $\vec{y}$.

The reasoning services that are of interest in *DL-Lite* are:

- *query-answering*: given a query $q(\vec{x})$ with distinguished variables $\vec{x}$ and a knowledge base $\mathcal{K}$, return all tuples $\vec{t}$ of objects that substituted to $\vec{x}$ are such that $\mathcal{K} \models q(\vec{t})$. Observe that as a special case of query answering we have concept satisfiability and logical implication of ABox assertions.

- *query-containment*: given two queries $q_1$ and $q_2$ and a knowledge base $\mathcal{K}$, verify whether $\mathcal{K} \models q_1 \sqsubseteq q_2$, i.e., whether in every model $\mathcal{I}$ of $\mathcal{K}$ the tuples of objects

that form the extension of $q_1$ in $\mathcal{I}$ are also in the extension of $q_2$ in $\mathcal{I}$. Observe that as a special case of query containment we have logical implication of inclusion assertions involving atomic concepts on both sides.

In fact, it can be shown that query containment can be reformulated as query answering, in particular with the help of soft constants. Hence, when we discuss reasoning (see Section 4) we will focus on query answering only.

## 3 Why *DL-Lite* is a "rich" DL

Although equipped with advanced reasoning services, at first sight *DL-Lite* seems to be rather weak in modeling intensional knowledge, and hence of limited use in practice. In fact this is not the case. Despite the simplicity of its language and the specific form of inclusion assertions allowed, *DL-Lite* is able to capture the main notions (though not all, obviously) of conceptual modeling formalism used in databases and software engineering such as ER and UML class diagrams.

In particular, *DL-Lite* assertions allow us to specify (below $A$, $A_1$ and $A_2$ are atomic concepts, and $R$ is an atomic role):

- *ISA*, using assertions of the form $A_1 \sqsubseteq A_2$, stating that the class $A_1$ is a subclass of the class $A_2$;
- *class disjointness*, using assertions of the form $A_1 \sqsubseteq \neg A_2$, stating disjointness between the two classes $A_1$ and $A_2$;
- *role-typing*, using assertions of the form $\exists R \sqsubseteq A_1$ (resp., $\exists R^- \sqsubseteq A_2$), stating that the first (resp., second) component of the relation $R$ is of type $A_1$ (resp., $A_2$);
- *participation constraints*, using assertions of the form $A \sqsubseteq \exists R$ (resp., $A \sqsubseteq \exists R^-$), stating that instances of class $A$ participate to the relation $R$ as the first (resp., second) component;
- *non-participation constraints*, using assertions of the form $A \sqsubseteq \neg\exists R$ (resp., $A \sqsubseteq \neg\exists R^-$), stating that instances of class $A$ do not participate to the relation $R$ as the first (resp., second) component;
- *functionality restrictions*, using assertions of the form (funct $R$) (resp., (funct $R^-$)), stating that an object can be the first (resp., second) component of the relation $R$ at most once.

Notably two important modeling features are missing in *DL-Lite*:

- the ability of stating *covering constraints*, i.e., stating that each instance of a class must be an instance of (at least) one of its subclasses;
- the ability of stating subset constraints between relations.

These features are missing exactly to get the nice computational characteristics that we are after.

Instead, observe that the limitation to binary roles only is not crucial. Indeed, it is possible to extend the reasoning techniques reported here to $n$-ary relations without losing most nice computational properties.

Finally, let us comment on the ability of *DL-Lite* of asserting extensional knowledge using soft constants. These can be considered as an advanced form of *null values* stating that the object with the desired property exists, though its identifier is not known. In other words, soft constants act as existentially quantified variables whose scope is the entire knowledge base.

# 4 Query answering in *DL-Lite*

We now present an algorithm that computes the answers to a conjunctive query over a *DL-Lite* KB. In the following, for ease of exposition we assume that the input query is a boolean query: the extension of the algorithm to non-boolean queries is straightforward.

Due to space limitations, we are only able to provide an informal description of the algorithm; moreover, we assume that no soft constants are present in the ABox.

## 4.1 Algorithm

The algorithm takes as input a *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a boolean conjunctive query $q$, and returns a boolean value. The algorithm consists of five steps:

1. *TBox normalization*: inclusion assertions of $\mathcal{T}$ in which conjunctive concepts occur in the right-hand side are rewritten by iterative application of the rule: if $B \sqsubseteq C_1 \sqcap C_2$ occurs in $\mathcal{T}$, then replace this assertion in $\mathcal{T}$ with the two assertions $B \sqsubseteq C_1$ , $B \sqsubseteq C_2$. The normalized TBox resulting from such a transformation contains the following types of assertions:

   - *ISA assertions* of the form $A_1 \sqsubseteq A_2$, where $A_1$ and $A_2$ are atomic concepts;
   - *disjointness assertions* of the form $B_1 \sqsubseteq \neg B_2$ where $B_1$ is a basic concept (i.e., either an atomic or an existential concept) and $\neg B_2$ is a negated basic concept;
   - *role-typing assertions* of the form $\exists R \sqsubseteq B$ or $\exists R^- \sqsubseteq B$, where $B$ is a basic concept;
   - *participation assertions* of the form $A \sqsubseteq \exists R$ or $A \sqsubseteq \exists R^-$, where $A$ is an atomic concept;
   - *functionality assertions* of the form (funct $R$) or (funct $R^-$).

2. *KB consistency check*: this step checks whether the ABox $\mathcal{A}$ satisfies the functionality and disjointness assertions occurring in the TBox $\mathcal{T}$. Specifically:

(a) First, in order to check satisfiability w.r.t. disjointness assertions, the TBox is expanded by computing all the disjointness assertions implied by the inclusion assertions in $\mathcal{T}$. More precisely, the TBox $\mathcal{T}$ is closed with respect to the following inference rule: if the assertions $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_3$ occur in $\mathcal{T}$ (where $C_1, C_2, C_3$ are arbitrary concepts), then add the assertion $C_1 \sqsubseteq C_3$ to $\mathcal{T}$.

(b) Then, the algorithm checks satisfiability w.r.t. disjointness assertions in $\mathcal{T}$: e.g., the assertion $B_1 \sqsubseteq \neg B_2$ in $\mathcal{T}$ is satisfied in $\mathcal{A}$ iff there is no $a$ such that both $a : B_1$ and $a : B_2$ are in $\mathcal{A}$ (if $B_2$ is the existential concept $\exists R$ (resp., $\exists R^-$), then also assertions of the form $R(a, b)$ (resp., $R(b, a)$) are taken into account).

(c) Finally, also satisfiability of $\mathcal{A}$ w.r.t. functionality assertions is checked: e.g., the assertion (funct $R$) in $\mathcal{T}$ is satisfied in $\mathcal{A}$ iff there is no pair of assertions in $\mathcal{A}$ of the form $R(a, b)$, $R(a, c)$.

If there is a disjointness assertion or a functionality assertion in $\mathcal{T}$ that the ABox $\mathcal{A}$ does not satisfy, then the algorithm returns true (there is no model for the KB $\mathcal{K}$, therefore every query is trivially true), otherwise the algorithm executes the next step.

3. *Query expansion*: the conjunctive query is rewritten based on the ISA, role-typing, and participation assertions in $\mathcal{T}$. More specifically, starting from the initial conjunctive query, a union of conjunctive queries is computed, by essentially applying the ISA, role-typing, and participation assertions as concept rewriting rules, applied from right to left. For instance, in the presence of the ISA assertion $A \sqsubseteq C$, the query $C(a)$ can be rewritten as $A(a)$, while in the presence of the role-typing assertion $\exists R \sqsubseteq C$, the same query can be rewritten as $R(a, x)$, where $x$ is a new variable symbol. Intuitively, in expanding the query we essentially embed all the relevant knowledge of the TBox represented by ISA, role-typing, and participation assertions.

4. *Query evaluation*: Finally, the expanded query is evaluated over the ABox $\mathcal{A}$. More precisely, the algorithm returns true if and only if there exists a conjunct of the expanded query that has an image in the ABox. Basically, a conjunct has an image in the ABox $\mathcal{A}$ if there exists a substitution $\sigma$ from the variables occurring in the conjunct to the constants occurring in $\mathcal{A}$ such that for each atom $\phi$ occurring in the conjunct, $\sigma(\phi) \in \mathcal{A}$ (actually, if an existential concept occurs in the atom, then also role memberhip assertions can provide an image for the atom). In other words, the algorithm evaluates the union of conjunctive queries considering the ABox as a database.

### 4.2 Correctness

Informally, the correctness of the above reasoning technique is essentially due to the fact that the assertions in the TBox can be divided into two classes:

- disjointness and functionality assertions, taken into account by Step 2 of the algorithm;
- ISA, role-typing, and participation assertions, considered in Step 3.

Indeed, it can be shown that the interaction between these two classes of assertions is limited to the derivation of new disjointness assertions in the TBox closure computed during Step 2. After these steps, the TBox can be discarded in the final query evaluation step.

### 4.3 Complexity

As for the complexity of the algorithm, it is easy to prove that the algorithm runs in polynomial time with respect to the size of the knowledge base $\mathcal{K}$, while the computation time is exponential with respect to the size of the query. The latter is due to the fact that the union of conjunctive queries computed in Step 3 may consist of a number of disjuncts (each of polynomial size) that is exponential in the number of atoms in the body of the initial query. Moreover, the evaluation of each disjunct in Step 4 may take nondeterministic polynomial (i.e., for practical purposes, exponential) time in the number of atoms of the disjunct, and hence in the number of atoms in the body of the initial query.

The algorithm can be extended to the presence of soft constants in the ABox, by adding a unification step that takes into account the presence of functionality assertions on the soft constants. Such an extension does not affect the computational properties of the algorithm.

Finally, it can be shown that, in the presence of soft constants, containment between conjunctive queries can be immediately reduced to query answering by the well-known "query freezing" technique (see, e.g., [11]), in which soft constants are used to deal with possible equalities implied by functionality assertions.

Summarizing, the following property holds.

**Theorem 1** *Subsumption, query answering, and query containment in DL-Lite are polynomial in the size of the knowledge base.*

## 5 Conclusions

We have described *DL-Lite*, a new DL specifically tailored to capture conceptual data models and basic ontology languages, while keeping the worst-case complexity of sound and complete reasoning tractable.

In this paper we focused on binary roles only, but this is not a crucial limitation. Indeed, it is possible to extend the reasoning techniques reported here to $n$-ary relations without loosing their nice computational properties. On the other hand, the results reported in [5] imply that the introduction of subset constraints on roles (i.e., role inclusion assertions) makes our technique inapplicable.

**Acknowledgments** This research was partly supported by MIUR under FIRB (Fondo per gli Investimenti della Ricerca di Base) project "MAIS: Multichannel Adaptive Information Systems" in the context of the Workpackage 2 activities, and by the EU funded projects INFOMIX (IST-2001-33570) and SEWASIE (IST-2001-34825).

## References

[1] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[2] A. Borgida and R. J. Brachman. Conceptual modeling with description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 10, pages 349–372. Cambridge University Press, 2003.

[3] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003)*, pages 260–271, 2003.

[4] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. What to ask to a peer: Ontology-based query reformulation. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, 2004.

[6] 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.

[7] 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.

[8] 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 Publisher, 1998.

[9] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[10] 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.

[11] 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.

# Local tableaux for reasoning in distributed description logics *

Luciano Serafini
ITC-IRST, 38050 Povo, Trento, Italy
luciano.serafini@itc.it

Andrei Tamilin
DIT - University of Trento, 38050 Povo, Trento, Italy
andrei.tamilin@dit.unitn.it

**Abstract**

The last decade of basic research in the area of Description Logics (DL) has created a stable theory, efficient inference procedures, and has demonstrated a wide applicability of DL to knowledge representation and reasoning. The success of DL in the semantic web and the distributed nature of the last one inspired recently a proposal of Distributed DL framework (DDL). DDL is composed of a set of stand alone DLs pairwise interrelated with each other via collection of bridge rules. In this paper, we investigate the reasoning mechanisms in DDL and introduce a tableau-based reasoning algorithm for DDL, built on the top of the state of the art tableaux reasoners for DL. We also describe a first prototype implementation of the proposed algorithm.

## 1 Introduction

Ontologies have been advocated as the basic tools to support interoperability between distributed applications and web services. The basic idea is that different autonomously developed applications can meaningfully communicate by using a common repository of meaning, i.e. a shared ontology. The optimal solution obviously lies in having a unique worldwide shared ontology describing all possible domains. Unfortunately, this is unachievable in practice. The actual situation in the web is characterized by a proliferation of different ontologies. Each ontology describes a specific domain from different perspectives and at different level of granularity. The initial interoperability problem, therefore, passes from the application level to the ontology level. Though the semantic standardization is far to be reached, the syntactic standardization is almost there, as it is widely accepted that ontologies should be expressed in a language, which is a variation of a descriptive language [6, 8].

Given this situation, one of the challenges in the semantic web is of being able to deal with a large number of overlapping and heterogeneous *local ontologies*. We use the term "local" to stress the fact that each ontology describes a domain of interest from a local and subjective perspective. In this paper, we focus on the problem of

*We thank Chiara Ghidini and Floris Roelofsen for their feedback on this paper.

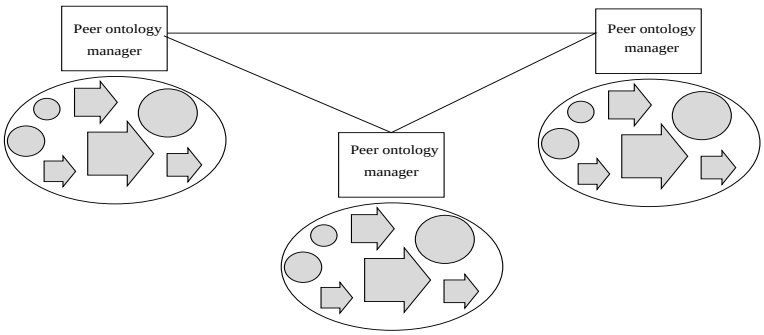



Figure 1: P2P architecture for managing multiple ontologies. In each peer, circles stand for ontologies, and arrows for semantic relations between ontologies.

*reasoning* within such web of local ontologies. We start form a long tradition of logics for distributes systems, based on propositional Multi-Context Systems [5, 4] and its Local Models Semantics [2], the extension of First Order Logics which leads to Distributed First Order Logics [3], and the extension of Description Logics which leads to Distributed Description Logics (DDL) [1]. Starting from these logical studies, our goal is to propose a *theoretically grounded* and *scalable* solution to the problem of reasoning with a set of distributed, heterogeneous, and overlapping local ontologies. Most of state of the art formalizations of that problem are based on a *global ontology* that allows to uniformly represent a set of local ontologies and semantic relations between them. In these approaches, reasoning in a set of local ontologies is rephrased into a problem of reasoning in the global ontology.

The approaches based on the global ontology, however, present two main drawbacks. First, from a computational complexity point of view it is more convenient to keep the reasoning as much local as possible, exploiting the structure provided by semantic relations for the propagation of reasoning through the local ontologies. Some intuition in this direction can be found in the computational complexity results for satisfiability in Multi-Context Systems described in [11]. Second, the reasoning procedure that has to be implemented in the global ontology should be capable of dealing with the most general local language, whereas having a more distributed approach would allow to apply to every local ontology the specific reasoner, optimized for the local language.

From the architectural point of view, our idea is inspired by peer-to-peer (P2P) distributed knowledge management architectures, proposed in the SWAP [13] and Edamok [12] projects, and by the C-OWL language [14]. We have implemented a P2P architecture, shown in Figure 1, consisting of peer ontology managers, providing reasoning services on a set of local ontologies, and capable of requesting reasoning services to other peers. The ontology manager of a pear $p$ is capable of providing *local* and *global* reasoning services. Local services involve only ontologies local to $p$, while global services involve both ontologies in $p$ and in other semantically related peers. Among the provided reasoning services, the fundamental ones are checking a local and a global subsumptions.

The paper contributes to the realization of the architecture described above with the following four points: (i) we describe a logical framework (DDL) capable of cap-

turing the behavior of the overall system, i.e. how subsumptions propagate through peers; (ii) give a general reference (and naïve) global algorithm for computing global subsumption, which is sound and complete w.r.t. any topology of the P2P ontology network; (iii) propose a distributed tableau algorithm for computing global subsumption, built as a composition of standard tableaux algorithms for computing local subsumption, which is sound and complete w.r.t. acyclic topology; and finally, (iv) describe a java-based prototype implementing the distributed tableau algorithm.

## 2 Distributed Description Logics

*Distributed description logics* (DDL), defined by Borgida and Serafini in [1], is a knowledge representation and reasoning formalism for describing distributed environments composed of a set of distinct description logics interrelated between each other through a set of pairwise inference connectives. In this section we briefly recall the definition of DDL as given in [1].

Before giving the formal definitions of DDL framework let us make several preliminary remarks. Given a non empty set $I$ of indexes, let $\{\mathcal{DL}_i\}_{i\in I}$ be a collection of description logics. Each $\mathcal{DL}_i$ can be one of the logics which is weaker or equivalent to $\mathcal{SHIQ}$ [9] (e.g. $\mathcal{ALC}$, $\mathcal{ALCN}$, $\mathcal{SH}$)[1]. For each $i \in I$ let us denote a T-box of $\mathcal{DL}_i$ as $\mathcal{T}_i$. To distinguish descriptions in each $\mathcal{DL}_i$, we will prefix them with the index of corresponding description logics. E.g. to reflect that any concept $C$ is stated locally in a terminology of $\mathcal{DL}_i$ we will write $i : C$; similarly, to reflect the fact that particular axiom, say $C \sqsubseteq D$, holds locally in a terminology of $\mathcal{DL}_i$ we will write $i : C \sqsubseteq D$.

*Bridge rules* are used to express semantic relations between different T-boxes.

**Definition 2.1 (Bridge rules).** A *bridge rule*, from $i$ to $j$ is an expression of the following two forms:

1. $i : x \xrightarrow{\sqsubseteq} j : y$, an *into-bridge rule*;
2. $i : x \xrightarrow{\sqsupseteq} j : y$, an *onto-bridge rule*;

where $x$ and $y$ are either two concepts, or two roles, or two individuals of $\mathcal{DL}_i$ and $\mathcal{DL}_j$ respectively.

In spite of this general definition, in this paper we concentrate on bridge rules between concepts. Intuitively, the into-bridge rule $i : C \xrightarrow{\sqsubseteq} j : D$ states that, from the $j$-th point of view the concept $C$ in $\mathcal{DL}_i$ is less general than its local concept $D$. Similarly, the onto-bridge rule $i : C \xrightarrow{\sqsupseteq} j : D$ expresses the fact that, according to $j$, $C$ in $\mathcal{DL}_i$ is more general than $D$ in $\mathcal{DL}_j$. Therefore, bridge rules from $i$ to $j$ represent the possibility of $\mathcal{DL}_j$ to translate (under some approximation) the foreign concepts of $\mathcal{DL}_i$ into its internal model. Note, that bridge rules are directional and reflect the subjective point of view of particular DL on other DLs surrounding it. Therefore, rules from $j$ to $i$ are not necessarily the inverse of rules from $i$ to $j$.

**Example 2.1.** The International Standard Classification of Occupations (ISCO-88)[2] is an ontology that organizes occupations in a hierarchical framework. At the lowest

[1] We assume familiarity with DLs and related tableaux-based systems described in [9].

[2] `http://www.ilo.org/public/english/bureau/stat/class/isco.htm`

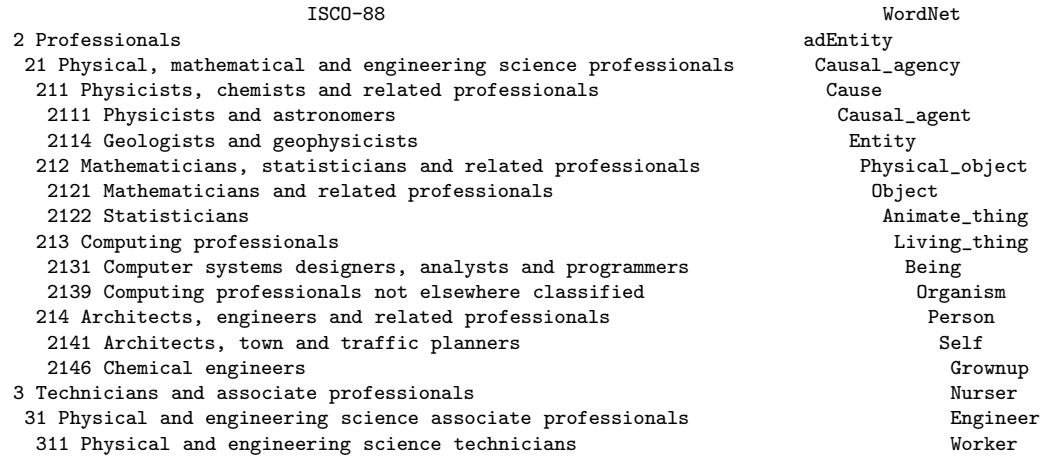



Figure 2: An extract from ISCO-88 and WordNet.

level is the unit of classification - a job - which is defined as a set of tasks or duties designed to be executed by one person. An extract of ISCO-88 is shown on the left side of Figure 2. A similar, though less detailed, ontology can be extracted from the People sub-hierarchy of WordNet[3]. Notice, that in WordNet there is no hierarchical classification of jobs, as the term "worker" is at the same level than "engineer". If, for whatever reason, one wants to import the ISCO-88 classification into WordNet, an example of bridge rules would be the following:

$$\text{ISCO} : \mathsf{Professionals} \xrightarrow{\sqsubseteq} \text{WNP} : \mathsf{Worker} \quad (1)$$

$$\text{ISCO} : \mathsf{Technicians\_And\_Associate\_Professionals} \xrightarrow{\sqsubseteq} \text{WNP} : \mathsf{Worker} \quad (2)$$

$$\text{ISCO} : \mathsf{Architects\_Engineers\_And\_Related\_Professionals} \sqcup \mathsf{Physical\_And\_Engineering\_Science\_Associate\_Professionals} \xrightarrow{\sqsupseteq} \text{WNP} : \mathsf{Engineer} \quad (3)$$

$$\text{ISCO} : \top \xrightarrow{\sqsubseteq} \text{WNP} : \neg\mathsf{Child} \quad (4)$$

$$\text{ISCO} : \mathsf{Doorkeepers\_watchpersons\_and}\ldots \xrightarrow{\sqsupseteq} \text{WNP} : \mathsf{Gatekeeper} \quad (5)$$

**Definition 2.2 (Distributed T-box).** A *distributed T-box (DTB)* $\mathfrak{T} = \langle \{\mathcal{T}_i\}_{i\in I}, \mathfrak{B}\rangle$ consists of a collection of T-boxes $\{\mathcal{T}_i\}_{i\in I}$, and a collection of bridge rules $\mathfrak{B} = \{\mathfrak{B}_{ij}\}_{i\neq j\in I}$ between them.

In order to deal with ontologies which are locally unsatisfiable (this can happen when a set of local axioms are not satisfiable or when bridge rules with other ontologies are not satisfiable) we will introduce two special types of local interpretations, called *holes*.

**Definition 2.3 (Holes).** A *full hole* in a T-box $\mathcal{T}$ is an interpretation $\mathcal{I}^\Delta = \langle \Delta^\mathcal{I}, \cdot^{\mathcal{I}^\Delta}\rangle$, where $\Delta^\mathcal{I}$ is the original nonempty domain in $\mathcal{T}$, and $\cdot^{\mathcal{I}^\Delta}$ is a function that maps every concept expression in $\mathcal{T}$ in the whole $\Delta^\mathcal{I}$. An *empty hole* in $\mathcal{T}$ as an interpretation $\mathcal{I}^\emptyset = \langle \Delta^\mathcal{I}, \cdot^{\mathcal{I}^\emptyset}\rangle$, where $\Delta^\mathcal{I}$ is the original nonempty domain $\mathcal{T}$, and $\cdot^{\mathcal{I}^\emptyset}$ is a function that maps every concept expression in $\mathcal{T}$ in the empty set.

[3] `http://xmlns.com/wordnet/1.6/Person`

According to the above definition, holes interpret every concept, both atomic and complex ones, either in the empty set or in the universe. The recursive definition of the interpretation of a concept does not apply for holes. One should notice that the interpretation of the concepts $(\neg C)$, denoted as $(\neg C)^{\mathcal{I}_\emptyset}$, is not $\Delta^{\mathcal{I}_\emptyset} \setminus C^{\mathcal{I}_\emptyset} = \Delta^{\mathcal{I}_\emptyset}$, but is $\emptyset$. The consequence of this fact is that $\mathcal{I}_\emptyset \models C \sqsubseteq D$ and $\mathcal{I}_\Delta \models C \sqsubseteq D$ for any pair of concepts $C$ and $D$. Obviously, since both $\mathcal{I}^\Delta$ and $\mathcal{I}^\emptyset$ satisfy all (even contradictory) concepts in $\mathcal{T}$, they are models of $\mathcal{T}$, i.e. $\mathcal{I}^\Delta \models \mathcal{T}$ and $\mathcal{I}^\emptyset \models \mathcal{T}$. Holes represent interpretations of locally inconsistent T-boxes.

**Definition 2.4 (Domain relation).** A *domain relation* $r_{ij}$ from $\Delta^{\mathcal{I}_i}$ to $\Delta^{\mathcal{I}_j}$ is a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$. We use $r_{ij}(d)$ to denote $\{d' \in \Delta^{\mathcal{I}_j} \mid \langle d, d' \rangle \in r_{ij}\}$; for any subset $D$ of $\Delta^{\mathcal{I}_i}$, we use $r_{ij}(D)$ to denote $\bigcup_{d \in D} r_{ij}(d)$; for any $R \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$ we use $r_{ij}(R)$ to denote $\bigcup_{\langle d,d' \rangle \in R} r_{ij}(d) \times r_{ij}(d')$.

A domain relation $r_{ij}$ represents the capability of $\mathcal{T}_j$ to map the elements of $\Delta^{\mathcal{I}_i}$ into its domain $\Delta^{\mathcal{I}_j}$. For instance if $\mathsf{John} \in \Delta^{\mathcal{I}_1}$ is a person and $\mathsf{J12} \in \Delta^{\mathcal{I}_2}$ is an individual that represents the student John in a specific school, the pair $\langle \mathsf{John}, \mathsf{J12} \rangle$ will be contained in $r_{12}$. Notice that $r_{12}$ is not necessarily a function. Indeed, John could attend two schools, and therefore, correspond to two individuals in $\Delta^{\mathcal{I}_2}$.

**Definition 2.5 (Distributed interpretation).** A *distributed interpretation* $\mathfrak{I} = \langle \{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I} \rangle$ of distributed T-box $\mathfrak{T}$ consists of local interpretations $\mathcal{I}_i$ on local domains $\Delta^{\mathcal{I}_i}$ for all $\mathcal{T}_i$, and a family of domain relations $r_{ij}$ between these local domains.

**Definition 2.6.** A distributed interpretation $\mathfrak{I}$ satisfies (written $\mathfrak{I} \models_d$) the elements of a DTB $\mathfrak{T}$ according to the following clauses: for every $i, j \in I$

1. $\mathfrak{I} \models_d i : A \sqsubseteq B$, if $\mathcal{I}_i \models A \sqsubseteq B$;
2. $\mathfrak{I} \models_d \mathcal{T}_i$, if $\mathfrak{I} \models_d i : A \sqsubseteq B$ for all $A \sqsubseteq B$ in $\mathcal{T}_i$;
3. $\mathfrak{I} \models_d i : x \xrightarrow{\sqsubseteq} j : y$, if $r_{ij}(x^{\mathcal{I}_i}) \subseteq y^{\mathcal{I}_j}$;
4. $\mathfrak{I} \models_d i : x \xrightarrow{\sqsupseteq} j : y$, if $r_{ij}(x^{\mathcal{I}_i}) \supseteq y^{\mathcal{I}_j}$;
5. $\mathfrak{I} \models_d \mathfrak{B}_{ij}$, if $\mathfrak{I}$ satisfies all bridge rules in $\mathfrak{B}_{ij}$;
6. $\mathfrak{I} \models_d \mathfrak{T}$, if for every $i, j \in I$, $\mathfrak{I} \models_d \mathcal{T}_i$ and $\mathfrak{I} \models_d \mathfrak{B}_{ij}$;
7. $\mathfrak{T} \models_d i : C \sqsubseteq D$ if for every $\mathfrak{I}$, $\mathfrak{I} \models_d \mathfrak{T}$ implies $\mathfrak{I} \models_d i : C \sqsubseteq D$.

Let us see now how bridge rules affect concept subsumption. Hereafter, $\mathfrak{B}_{ij}^{\text{into}}$ and $\mathfrak{B}_{ij}^{\text{onto}}$ will denote the set of into- and onto-bridge rules of $\mathfrak{B}_{ij}$ respectively.

**Monotonicity** Bridge rules do not delete local subsumptions. Formally:

$$\mathcal{T}_i \models A \sqsubseteq B \implies \mathfrak{T} \models_d i : A \sqsubseteq B \tag{6}$$

**Directionality** T-box without incoming bridge rules is not affected by other T-boxes. Formally, if $\mathfrak{B}_{ki} = \emptyset$ for any $k \neq i \in I$, then:

$$\mathfrak{T} \models_d i : A \sqsubseteq B \implies \mathcal{T}_i \models A \sqsubseteq B \tag{7}$$

**Strong directionality** Sole into- or onto-bridge rules incoming to local terminology do not affect it. Formally, if for all $k \neq i$ either $\mathfrak{B}_{ki}^{\text{into}} = \emptyset$ or $\mathfrak{B}_{ki}^{\text{onto}} = \emptyset$, then:

$$\mathfrak{T} \models_d i : A \sqsubseteq B \implies \mathcal{T}_i \models A \sqsubseteq B \tag{8}$$

**Local inconsistency** The fact that $\mathfrak{B}_{ij}$ contains into- and onto-bridge rules does not imply that inconsistency propagates. Formally:

$$\mathfrak{T} \models_d i : \top \sqsubseteq \bot \not\implies \mathfrak{T} \models_d j : \top \sqsubseteq \bot \tag{9}$$

**Simple subsumption propagation** Combination of onto- and into-bridge rules allows to propagate subsumptions across ontologies. Formally, if $\mathfrak{B}_{ij}$ contains $i : A \xrightarrow{\sqsupseteq} j : G$ and $i : B \xrightarrow{\sqsubseteq} j : H$, then:

$$\mathfrak{T} \models_d i : A \sqsubseteq B \implies \mathfrak{T} \models_d j : G \sqsubseteq H \tag{10}$$

**Generalized subsumption propagation** If $\mathfrak{B}_{ij}$ contains $i : A \xrightarrow{\sqsupseteq} j : G$ and $i : B_k \xrightarrow{\sqsubseteq} j : H_k$ for $1 \leq k \leq n$, then:

$$\mathfrak{T} \models_d i : A \sqsubseteq \bigsqcup_{k=1}^{n} B_k \implies \mathfrak{T} \models_d j : G \sqsubseteq \bigsqcup_{k=1}^{n} H_k \tag{11}$$

Among the given properties, property (9) and property (11) play special roles. The first one is important as it allows us to explain how full and empty holes constitute "locally inconsistent interpretations". The second one is important as it constitutes the main reasoning step of the tableau algorithm proposed in the next section. The proofs of the above properties can be found in [1, 10].

**Example 2.2.** In the hierarchy WNP of the previous example there is no subsumption relation between Engineer and Worker. From bridge rules (1–3) and from the fact that in the ISCO-88 ontology the concept Architects_Engineers_And_Related_Professionals is a subclass of Professionals, it is impossible to infer that Engineers is a subclass of Worker, i.e. that in WNP $\mathsf{Engineers} \sqsubseteq \mathsf{Worker}$. Similarly, the bridge rules (4) and (5) allow to infer that WNP classes Gatekeeper and Child are disjoint, i.e. that $\text{WNP} : \mathsf{Gatekeeper} \sqcap \mathsf{Child} \sqsubseteq \bot$.

# 3 Distributed reasoning in DDL

The reasoning services one would like to have in the web of ontologies are the following:

**Local reasoning services** are all kind of reasoning services one wants to have for a local ontology. The adjective "local" indicates that these reasoning services consider a local ontology as a stand alone object (no bridge rules are taken into account). The fundamental local reasoning service is *local subsumption*, i.e. the fact that $\mathcal{T}_i \models C \sqsubseteq D$.

**Global reasoning services** are services which take into account local ontologies in the context of the whole ontology space. These services should allow to infer subsumption between concepts on the basis of bridge rules, as well as new bridge rules on the basis of the existing ones. In this paper, we will focus on the basic global reasoning service that computes *global subsumption*, i.e. the fact that $\mathfrak{T} \models_d i : C \sqsubseteq D$.

A first proposal for implementing global reasoning services in DDL is based on reduction of a DTB $\mathfrak{T}$ to an equivalent global T-box $\mathcal{T}_G$, such that subsumption in $\mathfrak{T}$ can be computed via subsumption in $\mathcal{T}_G$ (see [1] for the transformation details). In this approach, however, a DTB can not be trivially reduced to a single global T-box simply by indexing the concepts and roles with the T-box they occur in. Furthermore, the reformulation done, works in the limited case when all local T-boxes are consistent. We therefore, would like to investigate a more general decision procedure.

Our proposal consist in building a distributed tableau for DDL on top of state of the art DL tableaux, implemented in FaCT and DLP[7], RACER[15], Pellet, and other DL systems. Given a concept $C$, they generate a tableau of $C$, $\mathbf{Tab}(C)$. Subsumption between concepts $C$ and $D$ is performed by checking the presence of clashes in all the branches of $\mathbf{Tab}(C \sqcap \neg D)$.

To understand how local tableaux are combined in order to check global subsumption we first consider a limited case of DDL that is composed of only two T-boxes $\mathcal{T}_1$ and $\mathcal{T}_2$, and bridge rules of only one direction from 1 to 2. Though this is unrealistic limitation, it constitutes a mandatory step, from which one can generalize and build a procedure capable of dealing with complex DDL topologies. For the sake of simplicity, we assume the second premise that requires the atomicity of concepts involved into bridge rules. This restriction can be later relaxed, since any bridge rule involving a complex concept $i : C$, can be replaced with a bridge with a new atomic concept $i : A$ and by the addition of the definition $A \equiv C$ to $\mathcal{T}_i$.

**Example 3.1.** For a distributed T-box $\mathfrak{T}_{12} = \langle \mathcal{T}_1, \mathcal{T}_2, \mathfrak{B}_{12} \rangle$, suppose that $\mathcal{T}_1$ contains axioms $A_1 \sqsubseteq B_1$ and $A_2 \sqsubseteq B_2$, $\mathcal{T}_2$ does not contain any axiom, and that $\mathfrak{B}_{12}$ contains the following bridge rules:

$$1 : B_1 \xrightarrow{\sqsubseteq} 2 : H_1 \qquad 1 : B_2 \xrightarrow{\sqsubseteq} 2 : H_2 \tag{12}$$

$$1 : A_1 \xrightarrow{\sqsupseteq} 2 : G_1 \qquad 1 : A_2 \xrightarrow{\sqsupseteq} 2 : G_2 \tag{13}$$

Let us show that $\mathfrak{T}_{12} \models_d 2 : G_1 \sqcap G_2 \sqsubseteq H_1 \sqcap H_2$, i.e. that for any distributed interpretation $\mathfrak{I} = \langle \mathcal{I}_1, \mathcal{I}_2, r_{12} \rangle$, $(G_1 \sqcap G_2)^{\mathcal{I}_2} \subseteq (H_1 \sqcap H_2)^{\mathcal{I}_2}$.

1. Suppose that by contradiction there is an $x \in \Delta_2$ such that $x \in (G_1 \sqcap G_2)^{\mathcal{I}_2}$ and $x \notin (H_1 \sqcap H_2)^{\mathcal{I}_2}$.
2. Then $x \in G_1^{\mathcal{I}_2}$, $x \in G_2^{\mathcal{I}_2}$, and either $x \notin H_1^{\mathcal{I}_2}$ or $x \notin H_2^{\mathcal{I}_2}$.
3. Let us consider the case where $x \notin H_1^{\mathcal{I}_2}$. From the fact that $x \in G_1^{\mathcal{I}_2}$, by the bridge rule (13), there is $y \in \Delta_1$ with $\langle y, x \rangle \in r_{12}$, such that $y \in A_1^{\mathcal{I}_1}$.

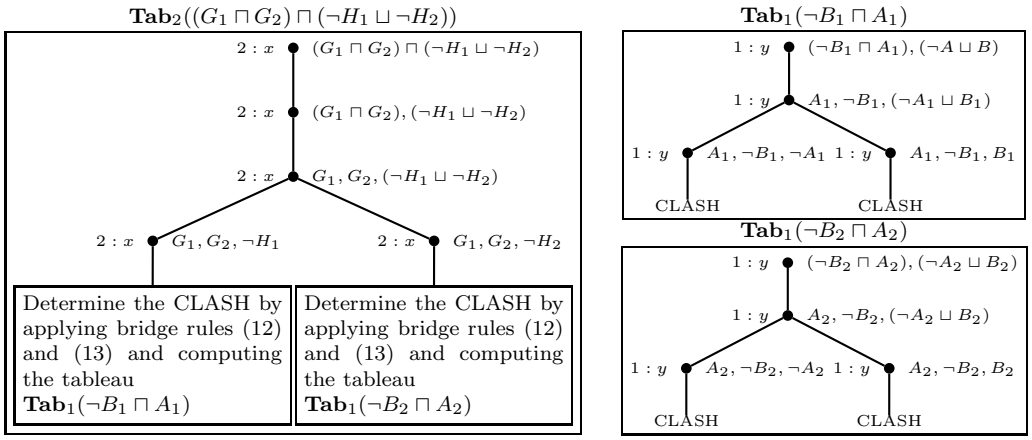



Figure 3: An example of distributed tableau.

4. From the fact that $x \notin H_1^{\mathcal{I}_1}$, by bridge rule (12), we can infer that for all $y \in \Delta_1$ if $\langle y, x \rangle \in r_{12}$ then $y \notin B_1^{\mathcal{I}_1}$.

5. But, since $A \sqsubseteq B \in \mathcal{T}_1$, then $y \in B_1^{\mathcal{I}_1}$, and this is a contradiction.

6. The case where $x \notin H_2^{\mathcal{I}_2}$ is similar.

The above combination of a tableau in $\mathcal{T}_2$ with a tableau in $\mathcal{T}_1$ gives a distributed tableau in $\mathfrak{T}$, depicted in Figure 3.

The intuitions given in Example 3.1 can be generalized for the case of multiple T-boxes, when there are no cyclical references between them. Formally, distributed T-box $\mathfrak{T} = \langle \{\mathcal{T}_i\}_{i \in I}, \{\mathfrak{B}_{ij}\}_{i \neq j \in I} \rangle$ is acyclical if $\mathfrak{B}_{ij} \neq \emptyset$ requires $i < j$ for all $i, j \in I$.

Algorithm 1 implements a distributed reasoning procedure intuitively introduced above. Here we define a distributed procedure **dTab**, which takes as an input a complex concept $\Phi$ to be verified and returns the result of its (un)satisfiability test. The algorithm first builds a local completion tree T by running local tableau algorithm **Tab**, and further attempts to close open branches of T by checking the bridge rules, which are capable of producing the clash in nodes of T. According to the local tableau algorithm, each node $x$ introduced during creation of the completion tree, is labeled with a function $L(x)$ containing concepts that $x$ must satisfy.

# 4 Prototype implementation

To evaluate the proposed distributed reasoning procedure we built a prototype modeling the P2P architecture given in Figure 1. Each peer ontology manager maintains ontologies in OWL and mappings in C-OWL, and provides local/global reasoning services, such as performing classification and checking entailment.

The key role in the ontology manager is played by a distributed reasoning engine, implementing developed distributed tableau algorithm. The kernel of the engine is formed by Pellet OWL DL reasoner[4]. Opennes of the source code and implementation

[4] `http://www.mindswap.org/2003/pellet`.

**Algorithm 1** Distributed reasoning procedure

**dTab**$_j(\Phi)$

1: BEGIN
2: T=**Tab**$_j(\Phi)$; {perform local reasoning and create completion tree}
3: **if** (T is not clashed) **then**
4: **for each** open branch $\beta$ in T **do**
5: **repeat**
6: select node $x \in \beta$ and an $i \neq j$;
7: $\mathbb{C}_i^{onto}(x) = \{C \mid i : C \xrightarrow{\sqsupseteq} j : D,\ D \in L(x)\}$;
8: $\mathbb{C}_i^{into}(x) = \{C \mid i : C \xrightarrow{\sqsubseteq} j : D, \neg D \in L(x)\}$;
9: **if** (($\mathbb{C}_i^{onto}(x) \neq \emptyset$) and ($\mathbb{C}_i^{into}(x) \neq \emptyset$)) **then**
10: **for each** $C \in \mathbb{C}^{onto}$ **do**
11: **if** (**dTab**$_i(C \sqcap \neg \bigsqcup \mathbb{C}_i^{into})$ is not satisfiable) **then**
12: close $\beta$; {clash in $x$}
13: break; {verify next branch}
14: **end if**
15: **end for**
16: **end if**
17: **until** (($\beta$ is open) and (there exist not verified nodes in $\beta$))
18: **end for**{all branches are verified}
19: **end if**
20: **if** (T is clashed) **then**
21: return unsatisfiable;
22: **else**
23: return satisfiable;
24: **end if**
25: END

in java made Pellet a good candidate for our prototype. Extension of the core reasoning functionality of Pellet transforms it to the distributed successor called *D-Pellet*.

To depicture the life cycle of D-Pellet, consider the case where a peer ontology manager is asked to perform one of the supported reasoning services in a local ontology it maintains. The ontology manager submits this query to D-Pellet, which in turn invokes the relative core Pellet functionality and checks for available mappings. Mapping processing can generate subqueries which are dispatched by the ontology manager to the corresponding foreign ontology manager. In turn, this starts another reasoning cycle. The reasoning stops when the initial D-Pellet receives the answers to the subproblems it sent out. Analysis of the subproblem answers defines the final reasoning result.

## 5 Conclusions

In this paper we have presented a tableau-based distributed reasoning procedure for DDL. We made several assumptions to study the reasoning in DDL, such as acyclicity of bridge rules and atomicity of concepts involved into bridge rules. The future work is to relax these assumptions in order to receive a practically usable framework.

## References

[1] A.Borgida and L.Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, pages 153–184, 2003.

[2] C.Ghidini and F.Giunchiglia. Local model semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.

[3] C.Ghidini and L.Serafini. Distributed first order logics. In *Proc. of the Frontiers of Combining Systems*, pages 121–139, 2000.

[4] F.Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993.

[5] F.Giunchiglia and L.Serafini. Multilanguage hierarchical logics (or: How we can do without modal logics). *Artificial Intelligence*, 65(1):29–70, 1994.

[6] G.Antoniou and F. van Harmelen. Web ontology language: Owl. In *Handbook on Ontologies in Information Systems*, pages 67–92, 2003.

[7] I.Horrocks and P.F.Patel-Schneider. FaCT and DLP. In *Proc. of the Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'98)*, pages 27–30, 1998.

[8] I.Horrocks, P.F.Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[9] I.Horrocks, U.Sattler, and S.Tobies. Practical reasoning for very expressive description logics. *Logic Journal of IGPL*, 8(3):239–263, 2000.

[10] L.Serafini and A.Tamilin. Distributed reasoning services for multiple ontologies. Technical Report DIT-04-029, University of Trento, 2004.

[11] L.Serafini and F.Roelofsen. Satisfiability for propositional contexts. In *Proc. of the Principles of Knowledge Representation and Reasoning (KR2004)*, 2004. Accepted for publication.

[12] M.Bonifacio, P.Bouquet, and P.Traverso. Enabling distributed knowledge management. Managerial and technological implications. *Novatica and Informatik/Informatique*, III(1), 2002.

[13] M.Ehrig, Ch.Tempich, J.Broekstra, F. van Harmelen, M.Sabou, R.Siebes, S.Staab, and H.Stuckenschmidt. A metadata model for semantics-based p2p systems. In *Proc. of the 2nd Konferenz Professionelles Wissensmanagement*, 2003.

[14] P.Bouquet, F.Giunchiglia, F. van Harmelen, L.Serafini, and H.Stuckenschmidt. C-owl: Contextualizing ontologies. In *Proc. of the 2d International Semantic Web Conference (ISWC2003)*, pages 164–179, 2003.

[15] V.Haarslev and R.Moller. Racer system description. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR2001)*, pages 701–706, 2001.

# A Description Logic Based Approach for Matching User Profiles

Andrea Calì[1,2], Diego Calvanese[2], Simona Colucci[3], Tommaso Di Noia[3] Francesco M. Donini[4]

[1]Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
I-00198 Roma, Italy
ac@andreacali.com

[2]Faculty of Computer Science
Free University of Bolzano/Bozen
Piazza Domenicani, 3
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

[3]Dip. di Elettrotecnica ed Elettronica
Politecnico di Bari
Via Re David 200
I-70125 Bari, Italy
{s.colucci,t.dinoia}@poliba.it

[4]Università della Tuscia
Facoltà di Scienze Politiche
Via San Carlo 32,
I-01100 Viterbo, Italy
donini@unitus.it

**Abstract**

Several applications require the matching of user profiles, e.g., job recruitment or dating systems. In this paper we present a logical framework for specifying user profiles that allows profile description to be incomplete in the parts that are unavailable or are considered irrelevant by the user. We present an algorithm for matching demands and supplies of profiles, taking into account incompleteness of profiles and incompatibility between demand and supply. We specialize our framework to dating services; however, the same techniques can be directly applied to several other contexts.

## 1 Introduction

The problem of matching demands and supplies of personal profiles arises in the business of recruitment agencies, in firms' internal job assignments, and in the recently emerging dating services. In all scenarios, a list of descriptions of persons is to be matched with a list of descriptions of required persons. In electronic commerce, the general problem is known as *matchmaking*, although here we do not consider any exchange of goods or services.

We stress the fact that in matchmaking, finding an exact match of profiles is not the objective; in fact, such a match is very unlikely to be found, and in all cases where an exact match does not exist, a solution to matchmaking must provide one or more *best possible* matches to be explored. Non-exact matches should consider both missing information — details that could be positively assessed in a second phase — and conflicting information — details that should be negotiated if the proposed match

is worth enough pursuing. Moreover, when several matches are possible, a matchmaker should list them in a most-promising order, so as to maximize the probability of a successful match within the first trials. However, such an order should be based on transparent criteria — possibly, logic — in order for the user to trust the system.

Profiles matchmaking can be addressed by a variety of techniques, ranging from simple bipartite graph matching (with or without cost minimization) [9], to vector-based techniques taken from classical Information Retrieval [11, 13, 12], to record matching in databases, among others. We now discuss some drawbacks of these techniques when transferred to solve matchmaking.

Algorithms for bipartite graph matching find optimal solutions when tying to maximize the number of matches [8, 10]. However, such algorithms rely on some way of assigning *costs* to every match between profiles. When costs are assigned manually, knowledge about them is completely implicit (and subjective), and difficult to revise. Moreover, in maximizing the number of matches a system may provide a *bad* service to single end users: for example, person $P_1$ could have a best match with job profile $J_1$, but she might be suggested to take job $J_2$ just because $J_1$ is the only available job for person $P_2$. Hence, from end user's viewpoint, maximizing the number of matches is not the feature that a matchmaker should have.

Both Database techniques for record matching (even with null values), and information retrieval techniques using similarity between weighted vectors of stemmed terms, are not suited for dealing with incomplete information usually present in matchmaking scenarios. In fact, information about profiles is almost always incomplete, not only because some information is unavailable, but also because some details are simply considered irrelevant by either the supplier or the demander — and should be left as such. Imposing a system interface for entering profiles with long and tedious forms to be filled in, is the most often adopted "solution" to this incompleteness problem — but we consider this more an escape for constraining real data into an available technique, than a real solution. For example, in a job posting/finding system, the nationality could be considered irrelevant for some profiles (and relevant for others); or in a dating service, some people may find disturbing (or simply inappropriate) the request to specify the kind of preferred music, etc. In such situations, missing information can be assumed as an "any-would-fit" assertion, and the system should cope with this incompleteness as is.

To sum up, we believe that there is a *representation problem* that undermines present solutions to matchmaking: considering how profiles information is represented is a fundamental step to reach an effective solution, and representations that are either too implicit, or overspecified, lead to unsatisfactory solutions.

Therefore, our research starts with proposing a language, borrowed from Artificial Intelligence, that allows for incomplete descriptions of profiles, and both positive and negative information about profiles. In particular, we propose a Description Logic [1] specifically tailored for describing profiles. Then, we model the matching process as a special reasoning service about profiles, along the lines of [5, 6]. Specifically, we consider separately conflicting details and missing details, and evaluate how likely is the match to succeed, given both missing and conflicting details. Our approach makes transparent the way matches are evaluated — allowing end users to request

justifications for suggested matches. We devise some special-purpose algorithms to solve the problem for the language we propose, and evaluate the possible application scenarios of a dating service.

The paper is organized as follows. In Section 2 we present the Description Logic we use for describing profiles. In Section 3 we describe how to represent user profiles, and in Section 4 we present the algorithm for matching user profiles. Section 5 concludes the paper.

## 2 A Description Logic for Representing Profiles

We use a restriction of the $\mathcal{ALC}(\mathcal{D})$ Description Logic, that, besides concepts and roles to represent properties of (abstract) objects, also allows one to express quantitative properties of objects, such as weight, length, etc., by means of *concrete domains* [2]. Each concrete domain $\mathcal{D}$, e.g., the real numbers $\mathbb{R}$, has a set of associated predicate names, where each predicate name $p$ denotes a predicate $p^{\mathcal{D}}$ over $\mathcal{D}$. For our purpose, it is sufficient to restrict the attention to unary predicates, and we assume that among such unary predicates we always have a predicate $\top$ denoting the entire domain, and predicates $\geqslant_\ell(\cdot)$ and $\leqslant_\ell(\cdot)$, for arbitrary values $\ell$ of $\mathcal{D}$. We also assume that the concrete domains we deal with are *admissible*, which is a quite natural assumption, satisfied e.g., by $\mathbb{R}$ (see [2] for the details). Besides roles, the logic makes use of *features*. Each feature has an associated concrete domain $\mathcal{D}$ and represents a (functional) relation between objects and values of $\mathcal{D}$.

Starting from a set of concept names (denoted by the letter $A$), a set of role names (denoted by $R$), a set of unary predicate names (denoted by $p$), and a set of features (denoted by $f$), we inductively define the set of *concepts* (denoted by $C$) as follows. Every concept name $A$ is a concept (atomic concept), and for $C_1$ and $C_2$ concepts, $R$ a role name, $f$ a feature with associated domain $\mathcal{D}$, and $p$ a unary predicate of $\mathcal{D}$, the following are concepts:

- $C_1 \sqcap C_2$ (conjunction), $C_1 \sqcup C_2$ (disjunction), and $\neg C$ (negation);
- $\exists R.C$ (existential restriction) and $\forall R.C$ (universal restriction);
- $p(f)$ (predicate restriction).

To express intentional knowledge about concepts, we make use of a *concept hierarchy*, which is a set of assertions of the form $A_1 \sqsubseteq A_2$ and $A_1 \sqsubseteq \neg A_2$, with $A_1$ and $A_2$ concept names. The former assertion expresses an *inclusion*, while the latter expresses a *disjointness*. For example, $\mathsf{football} \sqsubseteq \mathsf{sport}$ and $\mathsf{male} \sqsubseteq \neg\mathsf{female}$ could be assertions that are part of a concept hierarchy.

Formally, the semantics of concepts is defined by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consisting of an *abstract domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept name $A$ a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$; to each role name $R$ a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, and to each feature name $f$, associated with the concrete domain $\mathcal{D}$, a partial function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow \mathcal{D}$. The interpretation function can be extended to arbitrary concepts as follows:

$$(C_1 \sqcap C_2)^{\mathcal{I}} \;=\; C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$$

$$
\begin{aligned}
(C_1 \sqcup C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \neg C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{c \in \Delta^{\mathcal{I}} \mid \text{there exists } d \in \Delta^{\mathcal{I}} \text{ s.t. } (c,d) \in R^{\mathcal{I}} \text{ and } d \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{c \in \Delta^{\mathcal{I}} \mid \text{for all } d \in \Delta^{\mathcal{I}} \text{ s.t. } (c,d) \in R^{\mathcal{I}} \text{ we have } d \in C^{\mathcal{I}}\} \\
(p(f))^{\mathcal{I}} &= \{c \in \Delta^{\mathcal{I}} \mid f^{\mathcal{I}}(c) \in p^{\mathcal{D}}\}
\end{aligned}
$$

An assertion $A_1 \sqsubseteq A_2$ is *satisfied* by an interpretation $\mathcal{I}$ if $A_1^{\mathcal{I}} \subseteq A_2^{\mathcal{I}}$. An assertion $A_1 \sqsubseteq \neg A_2$ is satisfied by an interpretation $\mathcal{I}$ if $A_1^{\mathcal{I}} \cap A_2^{\mathcal{I}} = \emptyset$. We call an interpretation that satisfies all assertions in a hierarchy $\mathcal{H}$ a *model* of $\mathcal{H}$. A concept $C$ is *satisfiable in* $\mathcal{H}$ if $\mathcal{H}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. A hierarchy $\mathcal{H}$ *logically implies* an assertion $C_1 \sqsubseteq C_2$ between arbitrary concepts $C_1$ and $C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, for each model $\mathcal{I}$ of $\mathcal{H}$.

## 3 Representing User Profiles

We describe how to represent user profiles using the Description Logic presented in Section 2. The user profiles are tailored for dating services, though the same framework can be used, with small modifications, for different applications. We do not use the full expressive power of the Description Logic. In particular, we use a single role $\mathsf{hasInterest}$, to express interest in topics[1], and we make a limited use of the constructs. We assume the set of features to represent physical characteristics such as age, height, etc. Additionally, we use a special feature $\mathsf{level}$ that expresses the level of interest in a certain field. The concrete domain associated to $\mathsf{level}$ is the interval $\{\ell \in \mathbb{R} \mid 0 < \ell \leqslant 1\}$.

A user profile $P$ consists of the conjunction of the following parts:

- A conjunction of *atomic concepts*, to represent atomic properties associated to the user. We denote the set of such concepts as $\mathit{Names}(P)$.
- A conjunction of concepts of the form $p(f)$, to represent physical characteristics. The (unary) predicate $p$ can be one of the predicates $\geqslant_\ell(\cdot)$, $\leqslant_\ell(\cdot)$, $=_\ell(\cdot)$, where $\ell$ is a value of the concrete domain associated to $f$, or any logical conjunction of them. We denote the set of such concepts as $\mathit{Features}(P)$. Since $(p_1 \wedge p_2)(f)$ is equivalent to $p_1(f) \sqcap p_2(f)$, in the following, we can assume w.l.o.g. that $\mathit{Features}(P)$ contains at most one concept of the form $p(f)$ for each feature $f$.
- A conjunction of concepts of the form $\exists \mathsf{hasInterest}.(C \sqcap \geqslant_x(\mathsf{level}))$, where $C$ is a conjunction of concept names, and $0 \leqslant x \leqslant 1$. Each such concept represents an interest in a concept $C$ with level at least $x$. We denote the set of such concepts as $\mathit{Interests}(P)$.
- A conjunction of concepts of the form $\forall \mathsf{hasInterest}.(\neg C \sqcup \leqslant_x(\mathsf{level}))$, where $C$ is a conjunction of concept names, and $0 \leqslant x \leqslant 1$. Each such concept represents the

[1]For modeling profiles in different contexts, additional roles could be added to this language. For example, $\mathsf{hasSkill}$ for expressing skills in certain fields.

fact that the interest in a concept $C$ has level at most $x$. Note that, to represent the complete lack of interest in $C$, it is sufficient to put $x = 0$. We denote the set of such concepts as $\mathit{NoInterests}(P)$.

**Example 1** A supplied profile describing, say, a 35-years-old male, 1.82 cm tall, with strong interests in fantasy novels and japanese comics, fair interest in politics and no interest in football, could be expressed as follows:

$$\begin{array}{l}\mathsf{male} \sqcap {=_{35}}(\mathsf{age}) \sqcap {=_{1.82}}(\mathsf{height}) \sqcap \\ \exists \mathsf{hasInterest}.(\mathsf{fantasyNovels} \sqcap {\geqslant_{0.8}}(\mathsf{level})) \sqcap \\ \exists \mathsf{hasInterest}.(\mathsf{japaneseComics} \sqcap {\geqslant_{0.8}}(\mathsf{level})) \sqcap \\ \exists \mathsf{hasInterest}.(\mathsf{politics} \sqcap {\geqslant_{0.4}}(\mathsf{level})) \sqcap \\ \forall \mathsf{hasInterest}.(\neg \mathsf{football} \sqcup {\leqslant_{0}}(\mathsf{level}))\end{array}$$

where we suppose that interests are organized in a hierarchy including $\mathsf{fantasyNovels} \sqsubseteq \mathsf{novels}$, $\mathsf{japaneseComics} \sqsubseteq \mathsf{comics}$, and $\mathsf{male} \sqsubseteq \neg\mathsf{female}$

Observe that, when a profile is demanded, usually features like age and height will be used with range predicates (e.g., $(\geqslant 30 \wedge \leqslant 70)(\mathsf{age})$), instead of equality predicates as in the above example.

The following property follows immediately from the semantics of existential restriction. For every pair of concepts $C_1$ and $C_2$, role $R$, feature $f$ with associated concrete domain $\mathcal{D}$, and $p$ a predicate of $\mathcal{D}$:

$$\text{if } \mathcal{H} \models C_1 \sqsubseteq C_2 \quad \text{then} \quad \mathcal{H} \models \exists R.(C_1 \sqcap {\geqslant_{\ell}}(f)) \sqsubseteq \exists R.(C_2 \sqcap {\geqslant_{\ell}}(f))$$

For example, if $\mathsf{football} \sqsubseteq \mathsf{sport}$, then someone with a level of interest $\ell$ in football has at least the same level of interest in sport. This property is exploited in the matching algorithm provided in Section 4.

# 4 The Matching Algorithm

We present the algorithm for matching user profiles. The matching is performed over two profiles: the *demand* profile $P_d$ and the *supply* profile $P_s$. The algorithm is not symmetric, i.e., it evaluates how $P_s$ is suited for $P_d$, which is different from how $P_d$ is suited for $P_s$ [7]; of course, in order to determine how $P_d$ is suited for $P_s$, we can simply exchange the arguments of the algorithm.

From a logical point of view, we extend the non-standard inferences *contraction* and *abduction* defined in [4]. In particular, our contraction either removes or weakens conjuncts from $P_d$ so as to make $P_d \sqcap P_s$ satisfiable in $\mathcal{H}$; abduction, instead, either adds or strengthens conjuncts in $P_s$ so as to make $\mathcal{H} \models P_s \sqsubseteq P_d$. The algorithm is based on structural algorithms for satisfiability and subsumption [3]. Since it is reasonable to assume that users do not enter contradicting information, we assume that the profiles $P_d$ and $P_s$ are consistent.

The result of the match is a *penalty* in $\mathbb{R}$: the larger the penalty, the less $P_s$ is suited for $P_d$. In particular, partial penalties are added to the overall penalty by matching corresponding conjuncts of the two profiles; this is done in two ways.

**Contraction.** When a conjunct $C_d$ in $P_d$ is in contrast with some conjunct $C_s$ in $P_s$, then $C_d$ is removed and a penalty is added. Intuitively, since the supplier has something the demander does not like, in order to make the profiles match the demander *gives up* one of her requests. For example, let $C_d = \forall \mathsf{hasInterest}.(\neg \mathsf{sport} \sqcup \leqslant_{0.2}(\mathsf{level}))$ and $C_s = \exists \mathsf{hasInterest}.(\mathsf{football} \sqcap \geqslant_{0.4}(\mathsf{level}))$, where we have $\mathsf{football} \sqsubseteq \mathsf{sport}$ in $\mathcal{H}$. In this case the demander looks for someone who does not like sports very much, while the supplier likes football and therefore he likes sports. In this case, pursuing the match would require the demander to give up his/her request about sports, so the algorithm adds a penalty $\Pi_{c\ell}(0.4, 0.2)$ that depends on the gap between the lower bound (0.4) of the supply and the upper bound (0.2) of the demand. Similarly, for a feature $f$ with contrasting predicates $p_d$ and $p_s$, a penalty $\Pi_{cf}(p_d(f), p_s(f))$ is added to take into account the removal of $p_d(f)$ from $P_d$. In case a concept $A_d$ representing an atomic property has to be removed, the algorithm makes use of another penalty function $\Pi_c(\cdot)$, whose argument is the concept $A_d$.

**Abduction.** When a conjunct $c_d$ in $P_d$ has no corresponding conjunct in $P_s$, we add a suitable conjunct $c_s$ in $P_s$ that makes the profiles match, and add a corresponding penalty. Intuitively, the demander wants something which the supplier does not provide explicitly; in this case we assume that the supplier may or may not satisfy the demander's request, and as a consequence of this possibility of conflict we add a penalty. This is done by means of a penalty function $\Pi_a(\cdot)$, whose argument is a concept $C$, that takes into account the addition of $C$ to $P_s$. When the level of interest must be strengthened, we use a function $\Pi_{a\ell}(\cdot)$, that takes into account the gap between bounds. Similarly, a penalty function $\Pi_{af}(\cdot)$ takes into account the addition of features.

**Algorithm** CalculatePenalty
  **Input** demand profile $P_d$, supply profile $P_s$, concept hierarchy $\mathcal{H}$
  **Output** real value penalty $\geqslant 0$
  penalty := 0;
  // **Contraction**
  **foreach** $A_d \in \mathit{Names}(P_d)$ **do**
    **if** there exists $A_s \in \mathit{Names}(P_s)$
      such that $\mathcal{H} \models A_d \sqsubseteq \neg A_s$
    **then** remove $A_d$ from $P_d$
        penalty := penalty + $\Pi_c(A_d)$
  **foreach** $p_d(f) \in \mathit{Features}(P_d)$ **do**
    **if** there exists $p_s(f) \in \mathit{Features}(P_s)$
      such that $\exists x.p_d(x) \wedge p_s(x)$ is unsatisfiable in the domain associated to $f$
    **then** remove $p_d(f)$ from $P_d$
        penalty := penalty + $\Pi_{cf}(p_d(f), p_s(f))$
  **foreach** $\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_d}(\mathsf{level})) \in \mathit{Interests}(P_d)$ **do**
    **foreach** $\forall \mathsf{hasInterest}.(\neg C_s \sqcup \leqslant_{x_s}(\mathsf{level})) \in \mathit{NoInterests}(P_s)$ **do**
      **if** $\mathcal{H} \models C_d \sqsubseteq C_s$ **and** $x_d \geqslant x_s$

**then** replace $\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_d}(\mathsf{level}))$ in $P_d$
with $\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_s}(\mathsf{level}))$
penalty := penalty + $\Pi_{c\ell}(x_d, x_s)$

**foreach** $\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_d}(\mathsf{level})) \in \mathit{NoInterests}(P_d)$ **do**
**foreach** $\exists \mathsf{hasInterest}.(C_s \sqcap \geqslant_{x_s}(\mathsf{level})) \in \mathit{Interests}(P_s)$ **do**
**if** $\mathcal{H} \models C_s \sqsubseteq C_d$ **and** $x_d \leqslant x_s$
**then** replace $\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_d}(\mathsf{level}))$ in $P_d$
with $\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_s}(\mathsf{level}))$
penalty := penalty + $\Pi_{c\ell}(x_s, x_d)$

// **Abduction**

**foreach** $A_d \in \mathit{Names}(P_d)$ **do**
**if** there does not exist $A_s \in \mathit{Names}(P_s)$ such that $\mathcal{H} \models A_s \sqsubseteq A_d$
**then** add $A_d$ to $P_s$
penalty := penalty + $\Pi_a(A_d)$

**foreach** $p_d(f) \in \mathit{Features}(P_d)$ **do**
**if** there exist $p_s(f) \in \mathit{Features}(P_s)$
**then if** $\forall x.p_s(x) \Rightarrow p_d(x)$ is false in the domain associated to $f$
**then** add $p_d(f)$ to $P_s$
penalty := penalty + $\Pi_{af}(p_d(f), p_s(f))$
**else** add $p_d(f)$ to $P_s$
penalty := penalty + $\Pi_{af}(p_d(f), \top(f))$

**foreach** $\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_d}(\mathsf{level})) \in \mathit{Interests}(P_d)$ **do**
**if** there does not exist $\exists \mathsf{hasInterest}.(C_s \sqcap \geqslant_{x_s}(\mathsf{level})) \in \mathit{Interests}(P_s)$
such that $\mathcal{H} \models C_s \sqsubseteq C_d$ **and** $x_s \geqslant x_d$
**then if** there exists $\exists \mathsf{hasInterest}.(C_s \sqcap \geqslant_{x_s}(\mathsf{level})) \in \mathit{Interests}(P_s)$
such that $\mathcal{H} \models C_s \sqsubseteq C_d$
**then** let $\exists \mathsf{hasInterest}.(C_s \sqcap \geqslant_{x_s}(\mathsf{level}))$ be the concept in $\mathit{Interests}(P_s)$
with maximum $x_s$ among those for which $\mathcal{H} \models C_s \sqsubseteq C_d$ holds
penalty := penalty + $\Pi_{a\ell}(x_d, x_s)$
**else** penalty := penalty + $\Pi_a(\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_d}(\mathsf{level})))$
add $\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_d}(\mathsf{level}))$ to $P_s$

**foreach** $\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_d}(\mathsf{level})) \in \mathit{NoInterests}(P_d)$
**if** there does not exist $\forall \mathsf{hasInterest}.(\neg C_s \sqcup \leqslant_{x_s}(\mathsf{level})) \in \mathit{NoInterests}(P_s)$
such that $\mathcal{H} \models C_d \sqsubseteq C_s$ **and** $x_d \geqslant x_s$
**then if** there exists $\forall \mathsf{hasInterest}.(\neg C_s \sqcup \leqslant_{x_s}(\mathsf{level})) \in \mathit{NoInterests}(P_s)$
such that $\mathcal{H} \models C_d \sqsubseteq C_s$
**then** let $\forall \mathsf{hasInterest}.(\neg C_s \sqcup \leqslant_{x_s}(\mathsf{level}))$ be the concept in $\mathit{Interests}(P_s)$
with minimum $x_s$ among those for which $\mathcal{H} \models C_d \sqsubseteq C_s$ holds
penalty := penalty + $\Pi_{a\ell}(x_s, x_d)$
**else** penalty := penalty + $\Pi_a(\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_d}(\mathsf{level})))$
add $\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_d}(\mathsf{level}))$ to $P_s$

**return** penalty

The penalty functions used in the algorithm are defined as follows.

- For an atomic concept $A_d$, $\Pi_c(A_d)$ and $\Pi_a(A_d)$ depend solely from domain knowledge; for example, if the demander searches for a female while the supplier is a male, we are expected to associate a very high penalty to $\Pi_c(\mathsf{female})$ while removing $\mathsf{female}$ from $P_d$ in the contraction phase.

- Given a feature $f$ and the predicates $p_d(f)$, and $p_s(f)$, let $I_d$ and $I_s$ be the intervals associated to $p_d$ and $p_s$ respectively, and $G$ the gap between them; we define
$$\Pi_{cf}(p_d(f), p_s(f)) = \frac{|G|}{|I_d \cup I_s \cup G|}$$
In other words, the penalty is calculated by dividing the gap between $I_d$ and $I_s$ by the sum of the sizes of $I_d$, $I_s$, and $G$.

  For abduction we define (notice that, since $P_d^c \sqcap P_s$ is consistent, there is no gap $G$, and since $\forall x.p_s(x) \Rightarrow p_d(x)$ is false in the domain associated to $f$, we have that $|I_s| > 0$):
$$\Pi_{af}(p_d(f), p_s(f)) = \frac{|I_s \setminus I_d|}{|I_s|}$$

- Given $x_d, x_s \in [0,1]$, $\Pi_{cl}(x_d, x_s) = x_d - x_s$ and $\Pi_{a\ell}(x_d, x_s) = \frac{x_d - x_s}{1 - x_s}$.

- For $C_d = \sqcap_{i=1}^n A_i$, we define
$$\Pi_a(\exists \mathsf{hasInterest}.(C_d \sqcap \geqslant_{x_d}(\mathsf{level}))) = x_d \cdot \sum_{i=1}^{n} \Pi_a(A_i)$$
$$\Pi_a(\forall \mathsf{hasInterest}.(\neg C_d \sqcup \leqslant_{x_d}(\mathsf{level}))) = \frac{1 - x_d}{\sum_{i=1}^{n} \frac{1}{\Pi_a(A_i)}}$$

Note that only the penalty functions $\Pi_a(\cdot)$ and $\Pi_c(\cdot)$, when calculated on atomic concepts, rely on domain knowledge; all other penalty functions are defined based on the previous ones, and independently of other domain knowledge.

It is easy to check that all subsumption tests $\mathcal{H} \models C_1 \sqsubseteq C_2$ in the algorithm can be done in polynomial time in the size of $\mathcal{H}$, $C_1$, and $C_2$. Hence, it can be straightforwardly proved that the complexity of the algorithm is polynomial w.r.t. the size of the input.

$P_d = \mathsf{male} \sqcap >_{30}(\mathsf{age}) \sqcap >_{1.80}(\mathsf{height})$
$\sqcap \exists \mathsf{hasInterest}.(\mathsf{literature} \sqcap \geqslant_{0.5}(\mathsf{level}))$
$\sqcap \exists \mathsf{hasInterest}.(\mathsf{politics} \sqcap \geqslant_{0.4}(\mathsf{level}))$

$P_s = \mathsf{male} \sqcap =_{35}(\mathsf{age}) \sqcap =_{1.70}(\mathsf{height})$
$\sqcap \exists \mathsf{hasInterest}.(\mathsf{fantasyNovels} \sqcap \geqslant_{0.8}(\mathsf{level}))$
$\sqcap \exists \mathsf{hasInterest}.(\mathsf{japaneseComics} \sqcap \geqslant_{0.8}(\mathsf{level}))$
$\sqcap \forall \mathsf{hasInterest}.(\neg \mathsf{football} \sqcup \leq_0(\mathsf{level}))$

Figure 1: Formalization of profiles of Example 2

**Example 2** Let $P_d$ be the demand for a "*man over thirty, taller than 180 cm, with fair interest in literature and politics*" and $P_s$ the supplied profile describing a "*35 year-old male, 1.70 cm tall, with strong interest in fantasy novels and japanese comics and no interest in football*". Such profiles are formalized in Figure 1 w.r.t. a hierarchy $\mathcal{H}$ including $\mathsf{fantasyNovels} \sqsubseteq \mathsf{novels}$, $\mathsf{japaneseComics} \sqsubseteq \mathsf{comics}$, $\mathsf{comics} \sqsubseteq \mathsf{literature}$ and $\mathsf{novels} \sqsubseteq \mathsf{literature}$. The evaluation of the matching algorithm on $P_s$ and $P_d$ w.r.t. $\mathcal{H}$ returns a penalty value equal to $\Pi_{cf}(1.80, 1.70) + \Pi_a(\exists \mathsf{hasInterest}.(\mathsf{politics} \sqcap \geqslant_{0.4}(\mathsf{level})))$. The first term represents the need of giving up the $\mathsf{height}$ requirement in $P_d$ during the contraction phase, while the second one takes into account the addition of $\mathsf{politics}$ among $\mathit{Interests}(P_s)$ during the abduction phase.

The following theorem establishes the correctness of the above algorithm w.r.t. the computation of contraction and abduction. We denote with $P_d^c$ the profile $P_d$ after contraction, and with $P_s^a$ the profile $P_s$ after abduction.

**Theorem 3** *Given a concept hierarchy $\mathcal{H}$, a demand profile $P_d$, and a supply profile $P_s$, the following properties hold: (i) $P_d^c \sqcap P_s$ is satisfiable in $\mathcal{H}$; (ii) $P_s^a$ is satisfiable in $\mathcal{H}$; (iii) $\mathcal{H} \models P_d^c \sqsubseteq P_s^a$. (iv) there does not exist a profile $P_s'$ more general than $P_s^a$ (i.e., $\mathcal{H} \models P_s^a \sqsubseteq P_s'$ and $\mathcal{H} \not\models P_s' \sqsubseteq P_s$) such that $\mathcal{H} \models P_s' \sqsubseteq P_s$ and $\mathcal{H} \models P_s' \sqsubseteq P_d^c$.*

*Proof (sketch). (i)* The proof is by construction of a model $\mathcal{I}$ of $\mathcal{H}$ such that $(P_d^c \sqcap P_s)^{\mathcal{I}} \neq \emptyset$. *(ii)* Follows directly from *(i)*, since in the abduction step we add to $P_s$ conjuncts that are already in $P_d^c$. *(iii) and (iv)* Follow by construction of $P_s^a$, since exactly those conjuncts of $P_d^c$ that are not subsumed by $P_s$ in $\mathcal{H}$ have been included in $P_s^a$. By the fact that $\mathcal{H}$ consists only of inclusions and disjointness assertions between pairs of atomic concepts, it is indeed sufficient to consider pairs of concepts to check subsumption. □

## 5 Conclusions

In this paper we have addressed the problem of matching user profiles, when the demander's and supplier's profiles can have missing or conflicting information. In such a case, we have to take into account that the demander may need to give up some of her requests, and/or she may need to make assumptions on unspecified properties of the supplier's profile. We have proposed a DL-based framework for expressing user profiles in this setting, and a language suited for dating services. We have proposed an ad-hoc structural algorithm for matching profiles that, given a demander's and a supplier's profile, returns a penalty: the higher the penalty, the less the two profiles are compatible. As a future work, we want to test the algorithm in real cases with a prototype that is currently under development: we believe that promising applications of our techniques can be dating, recruitment, and service discovery systems.

**Acknowledgments** The first two authors were partly supported by MIUR under FIRB (Fondo per gli Investimenti della Ricerca di Base) project "MAIS: Multichannel Adaptive Information Systems" in the context of the Workpackage 2 activities.

## References


[1] 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.

[2] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of IJCAI'91*, pages 452–457, 1991.

[3] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[4] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. Concept abduction and contraction in description logics. In *Proc. of DL 2003*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-81/`, 2003.

[5] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, M. Mongiello, and M. Mottola. A formal approach to ontology-based semantic match of skills descriptions. *J. of Universal Computer Science*, Special issue on Skills Management, 2003.

[6] T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *Proc. of IJCAI 2003*, pages 337–342, 2003.

[7] T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. A system for principled matchmaking in an electronic marketplace. In *Proc. of WWW 2003*, pages 321–330, May 20–24 2003.

[8] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, 1986.

[9] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 1995.

[10] J. Kennington and Z. Wang. An empirical analysis of the dense assignment problem: Sequential and parallel implementations. *ORSA Journal on Computing*, 3(4):299–306, 1991.

[11] D. Kuokka and L. Harada. Integrating information via matchmaking. *J. of Intelligent Information Systems*, 6:261–279, 1996.

[12] K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic matchmaking among heterogeneus ssoftware agents in cyberspace. *Autonomous agents and multi-agent systems*, 5:173–203, 2002.

[13] D. Veit, J. P. Müller, M. Schneider, and B. Fiehn. Matchmaking for autonomous agents in electronic marketplaces. In *Proc. of AGENTS '01*, pages 65–66. ACM, 2001.

# Semantics driven support for query formulation

Paolo Dongilli, Enrico Franconi, and Sergio Tessaris

Free University of Bozen-Bolzano, Italy
<*lastname*>@inf.unibz.it

**Abstract**

In this paper we describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE (SEmantic Webs and AgentS in Integrated Economies) European IST project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The final purpose of the tool is to generate a conjunctive query ready to be executed by some evaluation engine associated to the information system.

## 1 Introduction

In this paper we describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE (SEmantic Webs and AgentS in Integrated Economies) European IST project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The final purpose of the tool is to generate a conjunctive query (or a non nested Select-Project-Join SQL query) ready to be executed by some evaluation engine associated to the information system.

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The ontology defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user's rich vocabulary. The user can exploit the ontology's vocabulary to formulate the query, and she/he is guided by such a richer vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of our proposal. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related with the lack of schema comprehension and so enabling her/him to easily formulate meaningful queries. Queries can be specified through an iterative refinement process supported by the ontology through intensional navigation. The user may specify her/his request

This work has been partially supported by the EU projects Sewasie, KnowledgeWeb, and Interop.

using generic terms, refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, giving instead an explicit meaning to a query and to its subparts through classification.

In the literature there are several approaches at providing intelligent visual query systems for relational or object oriented databases (see [10] for an extensive survey). However, to our knowledge, the work presented in this paper is among the first well-founded intelligent systems for query formulation support in the context of ontology-based query processing. The strength of our approach derives from the fact that the graphical and natural language representation of the queries is underpinned by a formal semantics provided by an ontology language. The use of an appropriate ontology language enables the system engineers to precisely describe the data sources, and their implicit data constraints, by means of a system global ontology (see [9]). The same ontology is leveraged by the query interface to support the user in the composition of the query, rather than relying on a less expressive logical schema. The underlying technology used by the query interface is based on the recent work on query containment under constraints (see [8; 16]).

The paper is organised as follows. Firstly we present the system w.r.t. user viewpoint, with the functionalities of the interface, then we describe the semantics and the reasoning services supporting the query interface. These include the query language expressiveness, the ontology support to the query formulation, and the natural language verbalisation issues. Finally, we discuss related work and we draw some conclusions.

## 2 Query interface: the user perspective

Initially the user is presented with a choice of different query scenarios which provide a meaningful starting point for the query construction. The interface guides the user in the construction of a query by means of a diagrammatic interface, which enables the generation of precise and unambiguous query expressions.

Query expressions are compositional, and their logical structure is not flat but tree shaped; i.e. a node with an arbitrary number of branches connecting to other nodes. This structure corresponds to the natural linguistic concepts of noun phrases with one or more propositional phrases. The latter can contain nested noun phrases themselves.

A query is composed by a list of terms coming from the ontology (classes); e.g. “Supplier” and “Multinational”. Branches are constituted by a property (attributes or associations) with its value restriction, which is a query expression itself; e.g. “selling on Italian market”, where “selling on” is an association, and “Italian market” is an ontology term.

The focus paradigm is central to the interface user experience: manipulation of the query is always restricted to a well defined, and visually delimited, subpart of the whole query (the *focus*). The compositional nature of the query language induces a natural navigation mechanism for moving the focus across the query expression (nodes of the corresponding tree). A constant feedback of the focus is provided on the interface by means of the kind of operations which are allowed. The system suggests only the operations which are “compatible” with the current query expression; in the sense that do not cause the query to be unsatisfiable. This is verified against the formal model describing the data sources.

One of the main requirements for the interface is that it must be accessed by any HTML browser, even in presence of restrictive firewalls. This constraints the its design, which overall appearance is shown in Figure 1. The interface is composed by three functional elements. The

first one (top part) shows a natural language representation of the query being composed, and the current focus. The second one is the query manipulation pane (bottom part) containing a diagram representing the focus and its terminological context, together with tools to specialise the query. Finally, a query result pane containing a table representing the result structure. The first two components are used to compose the query, while the third one is used to specify the data which should be retrieved from the data sources. Because of lack of space, in this paper we concentrate on the query building part. Therefore we wont discuss the query result pane, which allows the user to define the columns of a table which is going to organise the data from the query result.

**Query textual representation** The first component consists of a text box representing the query expression in a natural language fashion. The user selects subparts of the query for further refinement. The selection defines the current focus, which will be represented in the diagrams described in the following sections. The selected subexpression can be modified (refined or extended) by means of the query manipulation pane.

Although the query verbalisation does not provide accounts of the query structure, the system is aware of the nesting (and so is the user). The system provides the feedback on the nesting by means of navigation in the query expression when the user is interested in selecting a subpart of the query. When a node is selected, then the system automatically selects the whole subtree rooted at the node selected by the user.

It is important to stress that, although natural language is used as feedback to represent the query, this is used in generation mode only. Since the user does not write queries directly, there is no need to parse any natural language sentence or to resolve linguistic ambiguities.

**Query manipulation pane** The elements in the pane represent the current selection, and the operations allowed in its context. It is organised as a diagram showing the taxonomic context of the selection (the central part), and tools enabling the user to build the query expression.

The central part of the interface is occupied by the diagram allowing what we call *substitution by navigation*; i.e. the possibility of substituting the selected portion of the query with a more specific or more general terms.

The central part in the diagram shows the main term of the focus. While the surrounding terms are either more specific or more general w.r.t. the query expression *from the focus viewpoint*. For example, w.r.t. the query showed in Figure 1 with the focus on the first term ("Supplier"), the terms "Merchant" and "Agent" are more general term in the ontology, while "Retailer" and "Wholesaler" are more specific. By selecting one of these terms, the user can substitute the whole focus with the selected term. The purpose of the substitution group is twofold: it enables the replacement of the focus and it shows the position of the selection w.r.t. the terms in the ontology.

It can be the case that in the ontology there are terms which are equivalent to the selected part. In this case the user is offered to replace the selection with the equivalent term by the activation of the `Replace Equivalent` button.

A different refinement enabled by the interface is by *compatible terms*. These are terms in the ontology whose overlap with the focus can be non-empty. These ontology terms can be added to the head of the selection by using the `Add Concept` pop-up menu. For example, "Student" is among the compatible terms for the focus "Employee", but "Textile" is not. The compatible terms are automatically suggested to the user by means of appropriate reasoning task on the ontology describing the data sources.

Analogously, the user can add properties to the focus: *associations* (e.g. “Industry with sector”), and/or *attributes* (e.g. “Employee whose name is”). This can be performed by means of a `Add Property` pop-up menu, which presents the possible alternatives. Name and value restrictions for each property are verbalised using meta information associated to the terms in the ontology. For example, the association “with sector” with the restriction “Textile” is shown as “belonging to the textile sector”.

Note that the terms and the properties proposed by the system depend on the overall query expression, not only on the focus. This means that subparts of the query expression, taken in isolation, would generate different suggestions w.r.t. those in their actual context in the query.

Sub-queries can be associated to new names by means of a `Define` button. This process corresponds to the definition of a new named view. These newly introduced names can be used to shorten the query expression, or as a simple mechanism to extend the ontology to build a customised user’s viewpoint.

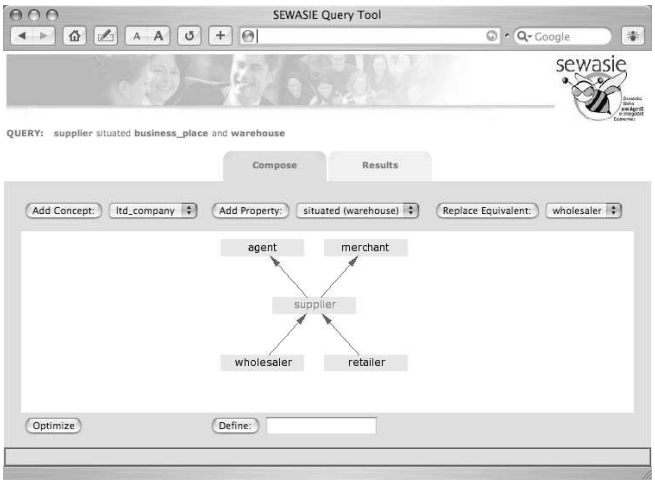



Figure 1: Query building interface.

# 3 Query interface: inside the box

In this section we describe the underpinning technologies and techniques enabling the user interface described in the previous sections. We will start by describing our assumptions on the query language, followed by system perspective over the described query building process. The whole system is supported by formally defined reasoning services which are described in Section 3.2. Finally, we introduce the verbalisation mechanism which enables the system to show the queries in a natural language fashion.

## 3.1 Conjunctive queries

Since the interface is build around the concept of classes and their properties, we consider conjunctive queries composed by unary (classes) and binary (attribute and associations) terms.

The body of a query can be considered as a graph in which variables (and constants) are nodes, and binary terms are edges. A query is connected (or acyclic) when for the corresponding graph the same property holds. Given the form of query expressions composed by the interface introduced in Section 2, we restrict ourselves to acyclic connected queries. This restriction is dictated by the requirement that the casual user must be comfortable with the language itself.[1] Note that the query language restrictions do not affect the ontology lan-

[1]Our technique can deal with disjunction of conjunctive queries, even with a limited form of negation applied to single terms. See [8; 16] for the technical details.

guage, where the terms are defined by a different (in our case more expressive) language. The complexity of the ontology language is left completely hidden to the user, who doesn't need to know anything about it.

To transform any query expression in a conjunctive query we proceed in a recursive fashion starting from the top level, and transforming each branch. A new variable is associated to each node: the list of ontology terms corresponds to the list of unary terms. For each branch, it is then added the binary query term corresponding to the property, and its restriction is recursively expanded in the same way.

Let us consider for example the query "Supplier and Multinational corporation selling on Italian market located in Europe", with the meaning that the supplier is located in Europe. Firstly, a new variable ($x_1$) is associated to the top level "Supplier and Multinational corporation". Assuming that the top level variable is by default part of the distinguished variables, the conjunctive query becomes

$$\{x_1 \mid \text{Suppl}(x_1), \text{Mult\_corp}(x_1), \ldots\},$$

where the dots mean that there is still part of the query to be expanded. Then we consider the property "selling on", with its value restriction "Italian market": this introduces a new variable $x_{1,1}$. The second branch is expanded in the same way generating the conjunctive query

$$\{x_1 \mid \text{Suppl}(x_1), \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}.$$

This transformation is bidirectional, so that a connected acyclic conjunctive query can be represented as a query expression (in the sense of Section 2) by dropping the variable names. As a matter of fact, the system is using this inverse transformation since the internal representation of queries is conjunctive queries.

Since a query is a tree, the focus corresponds to a selected sub-tree. It is easy to realise that each sub-tree is univocally identified by the variable corresponding to a node. Therefore, the focus is always on variable, and moving the focus corresponds to selecting a different variable. Modifying a query sub-part means operating on the corresponding sub-tree modifying the corresponding query tree.

*Substitution by navigation* corresponds to substitute the whole sub-tree with the chosen ontology term. The result would be a tree composed by a single node, without any branch, whose unary term is the given ontology term. In the *refinement by compatible terms*, the selected terms are simply added to the root node as unary query terms. For the *property extension*, adding an attribute or associations corresponds to the creation of a new branch. This operation introduces a new variable (i.e. node) with the corresponding restriction. When an attribute is selected, and a constant (or an expression) is specified, then this is added as restriction for the value of the variable.

### 3.2 Reasoning services and query interface

Reasoning services w.r.t. the ontology are used by the system to drive the query interface. In particular, they are used to discover the terms and properties (with their restrictions) which are proposed to the user to manipulate the query.

Our aim is to be as less restrictive as possible on the requirements for the ontology language. In this way, the same technology can be adopted for different frameworks, while the user is never exposed to the complexity (and peculiarities) of a particular ontology language.

In our context, an ontology is composed by a *set of predicates* (unary, binary), together with a *set of constraints* restricting the set of valid interpretations (i.e. databases) for the

predicates. The kind of constraints which can be expressed defines the expressiveness of the ontology language. Note that these assumptions are general enough to take account of widely used modelling formalisms, like UML for example.

We do not impose general restrictions on the expressiveness of the ontology language; however, we require the availability of two *decidable* reasoning services: satisfiability of a *conjunctive query*, and containment test of two conjunctive queries, both w.r.t. the constraints. If the query language includes the *empty* query (i.e. a query whose extension is always empty), then query containment is enough (a query is satisfiable iff it is not contained in the empty query). As described in Section 2, the query building interface represents the available operations on the query w.r.t. the current focus; i.e. the variable which is currently selected. Therefore, we need a way of describing a conjunctive query from the point of view of a single variable. The expression describing such a viewpoint is still a conjunctive query; which we call *focused*. This new query is equal to the original one, with the exception of the distinguished (i.e. free) variables: the only distinguished variable of the focused query is the variable representing the focus. In the following we represent as $q^x$ the query $q$ focused on the variable $x$. For example, the query
$q \equiv \{x_1, x_{1,2} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}$,
focused in the variable $x_{1,1}$ would simply be
$q^{x_{1,1}} \equiv \{x_{1,1} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}$.

The operations on the query expression require two different types of information: *hierarchical* (e.g. substitution by navigation), and on *compatibility* (e.g. refinement and new properties).

Let us consider the substitution by navigation with the more specific terms (the cases with more general and equivalent terms are analogous). Given the focused query $q^x$, we are interested to the unary atomic terms $T$ s.t. the query $\{y \mid T(y)\}$ is contained in $q^x$ and it is most general (i.e. there is no other query of that form contained in $q^x$, and containing $\{y \mid T(y)\}$).

Refinement by compatible terms and the addition of a new property to the query require the list of terms "compatible" with the given query. In terms of conjunctive queries, this corresponds to add a new term to the query. The term to be added should "join" with the query by means of the focused variable, and must be compatible in the sense that the resulting query should be satisfiable. This leads to the use of satisfiability reasoning service to check which predicates in the ontology are compatible with the current focus. With unary terms this check corresponds simply to the addition of the term $T(x)$ to the focused query $q^x$, and verify that the resulting query is satisfiable.

The addition of a property requires the discovery of both a binary term and its restriction: the terms to be added are of the form $\{x \mid R(x, y), T(y)\}$ if the focused variable is $x$. As for the refinement by compatible terms, the system should check all the different binary predicates from the ontology for their compatibility. This is practically performed by verifying the satisfiability of the query $q^x \bowtie \{x \mid R(x, y)\}$, for all atomic binary predicates $R$ in the signature and where $y$ is a variable not appearing in $q$.[2] Once a binary predicate $R$ is found to be compatible with the focused query, the restriction is selected as the most general unary predicate $T$ such that the query $q^x \bowtie \{x \mid R(x, y), T(y)\}$ is satisfiable.

[2] Here $\bowtie$ represents a natural join.

### 3.3 Using a Description Logics Reasoner

Although our approach is not tight to any ontology language, in the test implementation of our system we are using Description Logics (DLs). The reasons for this choice lie in the facts that DLs can capture a wide range of widespread modelling frameworks, and the availability of efficient and complete DL reasoners.

We adopted the Description Logics $\mathcal{SHIQ}$ (see [15]); which is expressive enough for our purposes, and for which there are state of the art reasoners. Note that the adoption of $\mathcal{SHIQ}$ allow us to use ontologies written in standard Web Ontology languages like OWL–DL (see [14]).

For space limitations we are not going to describe in detail the underlying $\mathcal{SHIQ}$ DL; the reader is referred to the above mentioned bibliographic references. The ontology contains unary (concepts) and binary (roles) predicates, and the constraints are expressed by means of inclusion axioms between concept or role expressions. One of the key features of $\mathcal{SHIQ}$ is the possibility of expressing the inverse of a role; which is extremely useful for converting tree–shaped queries into DL concept expressions.

Given the restriction to tree–shaped conjunctive query expressions, together with the availability of inverse roles, a focused query (see Section 3.2) corresponds to a concept expression (see [17]). Therefore, all the reasoning tasks described in Section 3.2 correspond to standard DL reasoning services. Again, this is not a restriction imposed by the underlying technology, since general conjunctive queries can be dealt with techniques described in [8; 16].

The idea behind the transformation of a query expression into a single concept description is very simple, and it is based on the fact that a concept expression can be seen as a query with a single distinguished variable. To focus the query on a variable, we start from the variable itself, then we traverse the query graph by encoding binary terms into DL existential restrictions and dropping the variable names. The fact that queries are tree–shaped ensures that variable names can be safely ignored. Let us consider for example the query expression

$$\{\text{Mult\_corp}(x_1), \text{Italian}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\}.$$

The DL expression corresponding to the query focused on $x_{1,1}$ is

$$(\text{It\_market} \sqcap \exists \text{sell\_on}^{-}(\text{Mult\_corp} \sqcap \text{Italian}));$$

where $\text{sell\_on}^{-}$ corresponds to the inverse of sell_on role.

As explained in Section 3.2, we need two kinds of information: hierarchical and compatibility. These, in the DL framework, are provided by the standard reasoning services of satisfiability and taxonomy position of a concept expression respectively. The first service verifies the satisfiability w.r.t. a knowledge base; while the second classifies a concept expression (i.e., provides it w.r.t. the ISA taxonomy of concept names).[3] Reasoning tasks described in Section 3.2 can be straightforwardly mapped into satisfiability and classification.

For example, checking the compatibility of the term Italian with the query

$$\{\text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\},$$

is performed by checking the satisfiability of the concept

$$\text{Italian} \sqcap \text{Mult\_corp} \sqcap \exists \text{sell\_onIt\_market}.$$

Compatibility of binary terms is performed analogously by using an existential restriction; e.g., $\exists \text{sell\_on}\top$.[4] To discover the restriction of a property we use classification instead of

[3] DL systems usually provide an efficient way of obtaining the taxonomic position of a given concept expression.

[4] Note the use of the $\top$ concept representing the whole domain (any possible concept).

repeated satisfiability. The idea is to classify the query focused on the variable introduced by the property. For example, to discover the restriction of sell_on applied to the query expression

$$\{x_1 \mid \text{Mult\_corp}(x_1), \text{Italian}(x_1)\},$$

we classify the expression $\exists \text{sell\_on}^{-}(\text{Mult\_corp} \sqcap \text{Italian})$. The DL reasoner returns the list of concept names more general and equivalent to the range of the relation sell_on, when restricted to the domain $(\text{Mult\_corp} \sqcap \text{Italian})$. This is exactly the information we need to discover the least general predicate(s) which can be applied to the property in the given context.

Our implementation uses the DL reasoner Racer (see [12]); which fully supports the $\mathcal{SHIQ}$ DL. The interaction with the DL reasoner is based on the DIG 1.0 interface API (see [1]), a standard to communicate with DL reasoners developed among different DL systems implementors. This choice makes our system independent from a particular DL reasoner, which can be substituted with any DIG based one.

### 3.4 Query verbalisation

The system always presents the user with a natural language transliteration of the conjunctive query. This is performed in an automatic way by using meta information associated with the ontology terms, both classes and properties. The verbalisation of the ontology terms must be provided in advance by the ontology engineers. For the verbalisation we use an approach similar to the one adopted by the Object Role Modelling framework (ORM, see [13; 19]).

Each class name in the ontology has associated a short noun phrase (usually one or two words), which represents the term in a natural language fashion. For example, to the class *PStudent* is associated "Postgraduate student" The user will see only the associated sentence, while *PStudent* is just used in the internal ontology representation.

For (binary) associations the ontology engineer has to provide two different verbalisations for the two directions. For example, let assume that the ontology states that the association *occ_room* links the two classes *PStudent* and *Room*. Then the engineer associates to the association the verbalisation "occupies" for the direction from *PStudent* to *Room*, and the verbalisation "is occupied by" for the other direction.

Attributes need one direction only, since they are never used from the point of view of the basic data type. In this case, the engineer is only required to provide the attribute verbalisation from the point of view of the class.

## 4 Discussion

The work proposed in this paper deals with a relatively new problem, namely providing the user with a visual interface to query heterogeneous data sources through an integrated ontology (that is, a set of constraints), and a specific literature does not exist yet. By looking at the extensive survey on Visual Query System (VQS) presented in [10] it easy to see that only little work has been done in the specific context we are dealing with. Some preliminary work was done by one research group [4; 11; 6; 5]. Similar work from the point of view of the visual interface paradigm, but without the well founded support of a logic-based semantics was carried out in the context of the Tambis project [18; 2]. Also [3] contains some interesting approach from the point of view of the visual interface, but again the system has a different background semantics.

In fact, only recently research has started to have a serious interest in query processing and information access supported by ontologies. Recent work has come up with proper semantics and with advanced reasoning techniques for query evaluation and rewriting using views under the constraints given by the ontology – also called view-based query processing [20; 7]. This means that the notion of accessing information through the navigation of an ontology modelling the information domain has its formal foundations.

This paper has presented the first well-founded intelligent user interface for query formulation support in the context of ontology-based query processing. This paper hopefully proved that our work has been done in a rigorous way both at the level of interface design and at the level of ontology-based support with latest generation logic-based ontology languages such as description logics, DAML+OIL and OWL. However, there are open problems and refinements which have still to be considered in our future work.

The system uses the verbalisations described in Section 3.4 to transform the conjunctive query into a natural language expression closer to the user understanding. In the course of the SEWASIE project some effort will be dedicated to explore semi-automatic techniques to rephrase the expressions in more succinct ways without loosing their semantic structure.

Another important aspect to be worked out is the understanding of the effective methodologies for query formulation in the framework of this tool, a task that needs a strong cooperation of the users in its validation. This will go in parallel with the interface user evaluation, which is just starting at the time of writing this paper.[5] The other crucial aspect is the efficiency and the scalability of the ontology reasoning for queries. We are currently experimenting the tool with various ontologies in order to identify possible bottlenecks.

We would like to thank Tiziana Catarci, Tania Di Mascio, and Giuseppe Santucci, for their valuable suggestions and discussions on the user interface. Moreover, the support of Ralf Möller and Volker Haarslev with the Racer reasoner has been essential for the development of our system prototype.

## References

[1] Sean Bechhofer, Ralf Mller, and Peter Crowther. The dig description logic interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.

[2] Sean Bechhofer, Robert Stevens, Gary Ng, Alex Jacoby, and Carole A. Goble. Guiding the user: An ontology driven interface. In *UIDIS 1999*, pages 158–161, 1999.

[3] Francesca Benzi, Dario Maio, and Stefano Rizzi. VISIONARY: a viewpoint-based visual language for querying relational databases. *J. Vis. Lang. Comput.*, 10(2):117–145, 1999.

[4] P. Bresciani and E. Franconi. Description logics for information access. In *Proceedings of the AI*IA 1996 Workshop on Access, Extraction and Integration of Knowledge*, Napoli, September 1996.

[5] Paolo Bresciani and Paolo Fontana. A knowledge-based query system for biological databases. In *Proceedings of FQAS 2002*, volume 2522 of *Lecture Notes in Computer Science*, pages 86–89. Springer Verlag, 2002.

[5] An on-line prototypical version of the query building tool, with a toy ontology without lexicalisation, is available at the URL `http://dev.eurac.edu:8090/sewasie/`.

[6] Paolo Bresciani, Michele Nori, and Nicola Pedot. A knowledge based paradigm for querying databases. In *Database and Expert Systems Application*, volume 1873 of *Lecture Notes in Computer Science*, pages 794–804. Springer Verlag, 2000.

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, 2000.

[8] Diego Calvanese, Giuseppe De Giacomo, and Maurizio 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.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS'98)*, pages 280–291, 1998.

[10] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[11] Enrico Franconi. Knowledge representation meets digital libraries. In *Proc. of the 1st DELOS (Network of Excellence on Digital Libraries) workshop on "Information Seeking, Searching and Querying in Digital Libraries"*, 2000.

[12] Volker Haarslev and Ralf Möller. Racer system description. In *Automated Reasoning: First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg, 2001.

[13] Terry A. Halpin. Augmenting UML with fact orientation. In *HICSS*, 2001.

[14] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.

[15] Ian Horrocks and Ulrike Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, pages 277–281, July 2002.

[16] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[17] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[18] Norman Murray, Carole Goble, and Norman Paton. A framework for describing visual interfaces to databases. *J. Vis. Lang. Comput.*, 9(4):429–456, 1998.

[19] `http://www.orm.net`, 2003.

[20] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf on Database Theory (ICDT'97)*, pages 19–40, 1997.

# From SHOQ(D) Toward $\mathcal{E}$-connections

Bernardo Cuenca Grau
Maryland Information and Network Dynamics Laboratory
University of Maryland, College Park, USA
bernardo@mindlab.umd.edu

Bijan Parsia
Maryland Information and Network Dynamics Laboratory
University of Maryland, College Park, USA
bparsia@isr.umd.edu

**Abstract**

In this paper, we propose a tableau-based technique for reasoning with various distributed DL knowledge bases. This technique can be applied both to DDLs and to new and relevant sublanguages of basic $\mathcal{E}$-connections. Its main advantage is that it is straightforward to implement by extending the existing tableau-based algorithms, as witnessed by our implementation in the Pellet OWL reasoner.

## 1 Introduction

Combining DL ontologies in a controlled and scalable way is crucial for the success of the Semantic Web. Recently, several proposals, like the Distributed Description Logics (DDL) [1] approach and the $\mathcal{E}$-connections framework [3] [4], have been presented as possible solutions for these and other related problems. In this paper, we define a new sub-formalism of basic $\mathcal{E}$-connections which is strictly more expressive than DDLs and that seems very straightforward to implement on existing tableau-based reasoners.

## 2 Perspectival $\mathcal{E}$-connections

Perspectival $\mathcal{E}$-connections (PECs) is an expressive sub-formalism of basic $\mathcal{E}$-connections which constraints the use of link properties in the component logics. For the simple case of two component logics, the set of links is partitioned into two disjoint sets $\epsilon = \epsilon_1 \cup \epsilon_2$, where $\epsilon_1 = \{E_j | j \in J\}, \epsilon_2 = \{F_k | k \in K\}$. The component logics are then enriched with the operators $< E_j >^1$, $< F_k >^2$. PECs are strictly less expressive than basic $\mathcal{E}$-connections because the use of the operators $< E_j >^2$ and $< F_k >^1$ is explicitly disallowed in the syntax and hence the links cannot be "navigated" in both directions. PECs are still strictly more expressive than DDLs

## 3 Reasoning technique

We have developed a tableaux-based reasoning technique for determining the satisfiability of concept terms in a certain PEC, whose component languages are DLs. The algorithm uses an instance of each tableaux-based decision procedure for the component DLs. In order to deal correctly with the new operators in the enriched language we need to define two new rules to each of the component decision procedures. These rules are basically analogous to the $\rightarrow \exists$ and $\rightarrow \forall$ rules in an ordinary tableau-based algorithm. For ensuring termination, a new blocking condition has to be defined

One important feature of this technique is that the decision procedures for the component logics are treated as black boxes in quite a similar way in which a DL reasoner considers a type checker as a black box when a DL is coupled to a conforming type system [2]. This shows that a slight modification of existing DL reasoners suffices for implementing the algorithm, as witnessed by our implementation in the Pellet OWL reasoner.

However, this technique cannot be straightforwardly extended to basic $\mathcal{E}$-connections. Intuitively, dealing with a link and its inverse breaks the black box condition and makes the algorithm unsound. Nominals also cause unsoundness if the algorithm is naively extended to PECs whose component logics contain nominals.

Finally, we have shown that this technique yields to a sound and complete algorithm for checking the satisfiability of concepts in a PEC, whose component languages are the SHIF DL or any of its sub-languages. Hence, we show that this technique can be used for combining OWL-Lite ontologies in the Semantic Web using the PEC formalism.

Future work includes the development of reasoning techniques for handling nominals in the combination (and hence OWL-DL ontologies), ABoxes, and also to explore the transition from PECs to full $\mathcal{E}$-connections. We are also looking into integrating support for multiple ontologies in the SWOOPed ontology editor in order to make these formalisms as usable and intuitive as possible for modelers, which is crucial for successfully bringing them to the Semantic Web.

## References


[1] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics, 1:153-184*, 2003.

[2] I. Horrocks and U. Sattler. Ontology reasoning in the shoq(d) description logic. In B. Nebel, editor, *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[3] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-connections of abstract description systems. *Artificial Intelligence*, 2003. To appear.

[4] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. $\mathcal{E}$-connections of description logics. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, CEUR-WS, 2003.

# Specifying the disjoint nature of object properties in DL

Cartik R. Kothari and David J. Russomanno
Department of Electrical and Computer Engineering
The University of Memphis, Memphis, TN 38152 USA
rkothari@memphis.edu, d-russomanno@memphis.edu

**Abstract**

This paper proposes constructs that can be used to declaratively specify the disjoint nature of object properties or roles. These constructs may be a useful extension to the Description Logic system that is the basis of OWL.

## 1 Introduction

The ability to specify disjoint relations has several applications in database and knowledge based systems. This paper introduces a set of syntactic constructs that can be used to specify the disjoint nature of roles in Description Logic (DL) [1] systems. Wessel [2] presents another study upon DL systems that specify the disjointness of roles; however, it did not specifically investigate the disjoint nature of roles. Instead, role disjointness was used as a starting premise to investigate the composition of roles in $ALC_{RA}$ DL, which was determined to be undecidable.

## 2 Specifying the disjoint nature of roles

The semantics of DL constructors is defined in terms of an interpretation $I = (\Delta^I, \cdot^I)$ that consists of a non-empty domain $\Delta^I$ and an interpretation function $\cdot^I$. The interpretation maps individual names (e.g., *x*, *y* and *z*) into objects or individuals of the domain; and the role names (e.g., *R1* and *R2*) into subsets of the Cartesian product of the domain ($\Delta^I \times \Delta^I$) as shown in (1) ~ (5).

$$x^I \in \Delta^I \quad (1)$$
$$y^I \in \Delta^I \quad (2)$$
$$z^I \in \Delta^I \quad (3)$$
$$R1^I \subseteq \Delta^I \times \Delta^I \quad (4)$$
$$R2^I \subseteq \Delta^I \times \Delta^I \quad (5)$$

Four types of role disjointness can now be distinguished as follows: 1) if an object appears as a range element in role *R1* then it cannot appear as a range element in *R2*; 2) if an object appears as a domain element in role *R1* then it cannot appear as a domain element in *R2*; 3) the conjunction of the conditions in 1 and 2; and 4) two roles can have no instances in common.

For the scenario in which two disjoint roles cannot have instances that have a common range object, the required semantics are shown in (6). A new construct ($|_r$) is proposed to capture the semantics of (6). The disjoint nature of *R1* and *R2* can now be specified as in (7). This constraint would not allow the same object to appear as the range in instances of both roles.

$$\forall x \forall y \forall z \, (x^I, y^I) \in R1^I \Rightarrow (z^I, y^I) \notin R2^I \quad (6)$$
$$R1 \,|_r\, R2 \quad (7)$$

For the scenario in which two disjoint roles cannot have instances that have a common domain object, the required semantics are shown in (8). A new construct ($|_d$) is proposed to capture the semantics of (8). The disjoint nature of *R1* and *R2* can now be specified as in (9). This

constraint does not permit the same object to appear as the domain in instances of both the roles *R1* and *R2*.

$$\forall x \forall y \forall z \, (x^I, y^I) \in R1^I \Rightarrow (x^I, z^I) \notin R2^I \quad (8)$$

$$R1 \mid_d R2 \quad (9)$$

The semantics of two disjoint roles such that no domain element in *R1* can appear as a domain element in *R2* and no range element in *R1* can appear as a range element of *R2* is shown in (10). A new construct (|) is defined to capture the semantics of (10). The disjoint nature of *R1* and *R2* can now be specified as in (11).

$$\forall x \forall y \forall z \forall w \, (x^I, y^I) \in R1^I \Rightarrow (x^I, z^I) \notin R2^I \wedge (w^I, y^I) \notin R2^I \quad (10)$$

$$R1 \mid R2 \quad (11)$$

Finally, each new construct ($|_r$, $|_d$, and |) expresses different semantics than rule (12), which states that two roles can have no instances in common as captured in (13). Applying the substitution {z/x} to (6) yields (12) as does applying the substitution {z/y} to (8). Rule (12) is a factor of (10) when applying the substitution {z/y, w/x} to (10). However, it is not possible with the semantics expressed in (12) alone to determine whether the two roles can share domain objects, range objects, or neither as captured by the new constructs.

$$\forall x \forall y \, (x^I, y^I) \in R1^I \Rightarrow (x^I, y^I) \notin R2^I \quad (12)$$

$$R1^I \cap R2^I = \perp \quad (13)$$

## 3 Conclusions

A premise of this paper is that Semantic Web knowledge representation formalisms should support the declarative representation of property disjointness. Four types of property disjointness have been described in this paper. It should be noted that if a knowledge engineering application required capturing the semantics provided by the constructs $|_r$, $|_d$, and |, a workaround could be declaratively achieved, albeit requiring minor re-conceptualization, to enforce the semantics. For example, to achieve $|_d$, the domain of roles *R1* and *R2* would be partitioned into two disjoint concepts and the disjoint nature of roles *R1* and *R2* would then be implied if they were re-defined to use these disjoint concepts as their respective, restricted domains. However, no such workaround appears to exist for expressing $R1 \cap R2 = \perp$ for roles defined on the same domain and range, which suggests that a *DisjointProperties*($R_1$ ... $R_n$) construct may be useful in Description Logics based ontology languages such as OWL. Role intersection constructs are provided by the *ALB* DL [3], which has been proved to be decidable. The analysis of the computational properties of the constructs proposed herein is the subject of ongoing investigation. The investigation will include decidability strategies discussed in [3] and its relevance to the proposed constructs.

## References


[1] D. Nardi and R. J. Brachman, "An Introduction to Description Logics," In F. Baader et al. (Eds.), *The Description Logic Handbook*, Cambridge University Press, 2003.

[2] M. Wessel, "Undecidability of $ALC_{RA}$," Technical Report No. FBI-HH-M-302/01, Computer Science Department, University of Hamburg, Germany, 2001.

[3] U. Hustadt and R. Schmidt, "Issues of Decidability for Description Logics in the Framework of Resolution," In R. Caferra and G. Salzer (Eds.), *Automated Deduction in Classical and Non-Classical Logics*, LNAI 1761, Springer, 2000, 192 – 206.

# Towards Explaining Semantic Matching

Deborah L. McGuinness[1] Pavel Shvaiko[2] Fausto Giunchiglia[2]
Paulo Pinheiro da Silva[1]

[1]Stanford University, Stanford, USA
{dlm,pp}@ksl.stanford.edu.
[2]University of Trento, Povo, Trento, Italy
{pavel,fausto}@dit.unitn.it

**Abstract**

Interoperability among systems using different term vocabularies requires mappings between them. Matching applications generate these mappings. When the matching process utilizes term meaning (instead of simply relying on syntax), we refer to the process as semantic matching. If users are to use the results of matching applications, they need information about the mappings. They need access to the sources that were used to determine relations between terms and potentially they need to understand how deductions are performed. In this paper, we discuss our approach to explaining semantic matching. Our initial work uses a satisfiability-based approach to determine subsumption and semantic matches and uses the Inference Web and its OWL encoding of the proof markup language to explain the mappings.

## 1 Semantic Matching

In this paper, we discuss semantic matching as introduced in [3], and implemented within the *S-Match* system [4]. We view information sources to be graph-like structures containing terms and their inter-relationships. The semantic matching distinguishes the following relations between terms: *equality* ($=$, mutual subsumption); *more general* ($\sqsupseteq$, subsumer); *less general* ($\sqsubseteq$, subsumee); *mismatch* ($\perp$, disjoint); *overlapping* ($\sqcap$, there may exist an instance of both classes). The semantic relations are calculated by mapping meaning which is codified in the element descriptions and the graphs in two steps: obtaining a representation of the node meaning and by determining the meaning of the node position in the graph. In order to obtain some information about the node labels, our initial implementation accesses WordNet. Extensions to the work would also take other DL representations of the classes as input such as full OWL ontologies. Semantic matching translates the matching problem into a validity check of the appropriate propositional formula. The algorithm then checks for sentence validity by proving that its negation is unsatisfiable. Our implementation uses the JSAT SAT reasoner.

## 2 Explaining Matching using Inference Web

Inference Web (IW) [6] enables applications to generate portable and distributed explanations for answers. In order to explain semantic matching and thereby increase

the trust level of its users, we need to provide information about background theories (initially Wordnet), the JSAT manipulations of sentences, and the semantic matching translations of graphs into propositional sentences. The IW proof and explanation documents are represented in PML [1] and are composed of PML *node sets*. This representation could be viewed as the web-ized distributed OWL version of one author's previous work on explaining description logics [7].

Users may need different types of explanations. For example, if negotiating agents trust each other's information sources, explanations should focus on the *S-Match* manipulations. If on the other hand, the sources may be suspect, explanations should focus on meta information about sources. If a user wants an explanation of the inference engine(s) embedded in a matching system, a more complex explanations may be required, see [9] for details. Our current version of *S-Match* uses JSAT, and in particular the Davis-Putnam-Longemann-Loveland (DPLL) procedure [2].

## 3 Discussion

While there are a number of other efforts in semi-automated schema/ontology matching [8], we are not aware that any provide explanations. By extending *S-Match* to use the IW infrastructure, we demonstrate our approach for explaining matching systems that use background ontological information and reasoning engines[1]. The DPLL procedure explained in our approach, while unoptimized, includes the essence of the state of the art SAT engines. Thus, one could consider using another optimized SAT reasoner that may be chosen for particular matching problems and use the approach discussed for generating explanations. Future work includes using more expressive background ontologies and other SAT engines as well as other non-SAT DPLL-based inference engines, e.g., DLP, FaCT [5].

## References


[1] P. Pinheiro da Silva, D. L. McGuinness, and R. Fikes. A proof markup language for semantic web services. TR KSL-04-01, Stanford University, 2004.

[2] M. Davis and H. Putnam. A computing procedure for quantification theory. In *Journal of the ACM*, number 7, pages 201–215, 1960.

[3] F. Giunchiglia and P. Shvaiko. Semantic matching. In *The Knowledge Engineering Review journal*, number 18(3), 2004. Also TR DIT-03-013.

[4] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS' 04*, 2004. Also TR DIT-04-015.

[5] I. Horrocks and P. F. Patel-Schneider. Fact and dlp. In *Automated Reasoning with Analytic Tableaux and Related Methods: Tableaux'98*, pages 27–30, 1998.

[6] D. L. McGuinness and P. Pinheiro da Silva. Infrastructure for web explanations. In *Proceedings of ISWC'03*, pages 113–129, 2003.

[7] D.L. McGuinness. *Explaining reasoning in description logics*. PhD thesis, Rutgers University, 1996.

[8] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. In *VLDB Journal*, number 10(4), pages 334–350, 2001.

[9] P. Shvaiko, F. Giunchiglia, P. Pinheiro da Silva, and D. L. McGuinness. Web explanations for semantic heterogeneity discovery. TR KSL-04-02, Stanford University, 2004.


[1] Long version of this paper is available at http://www.dit.unitn.it/research/publications/techRep?id=549 as TR DIT-04-019 and at http://www.ksl.stanford.edu/people/dlm/papers/dl04long-abstract.html

# Extending DL Reasoning Support for the OWL Datatyping (or "Why Datatype Groups?")

Jeff Z. Pan and Ian Horrocks
Department of Computer Science,
University of Manchester, UK M13 9PL
{last-name}@cs.man.ac.uk

The OWL [2] datatype formalism (or simply *OWL datatyping*) presents some new requirements for DL reasoning services, in terms of semantics (to allow the use of so-called 'un-supported' datatypes), expressive power (to support enumerated datatypes) and datatype construction mechanism (both datatypes and datatype expressions). On the other hand, OWL datatyping is expected to be extended to include more expressive power. E.g., OWL datatyping does not provide a general framework for user-defined datatypes, such as XML Schema derived datatypes, nor does it support $n$-ary datatype predicates (such as the binary predicate $>$ for integers), not to mention user-defined datatype predicates (such as the binary predicate $>$ for non-negative integers). In this poster, we explain why it is necessary to extend the existing datatype approaches to the datatype group approach, in order to meet the above new requirements.

It was Baader and Hanschke [1] who first presented a rigorous treatment of datatype predicates (or simply *predicates*). In their approach, a concrete domain [1, 4] is composed of a set of datatype values (such as integers) and a set of *n-ary* predicates (such as '$<$') defined over these values with obvious (fixed) extensions. Horrocks and Sattler [3] proposed the so called 'type system approach', which can be seen as a simplified version of the concrete domain approach, where the datatype domain (of a datatype interpretation) is regarded as a universal concrete domain and datatypes are treated as unary predicates in the universal concrete domain. In short, in the above two approaches, datatypes are nothing but unary predicates.

In OWL datatyping, however, people take another view. A Datatype $d$ distinguishes from a predicate in that it is characterised not only by the value spaces $V(d)$, but also a lexical space, $L(d)$, which is a set of Unicode strings, and a total mapping $L2V(d)$ from the lexical space to the value space. E.g., *boolean* is a datatype with value space $\{true, false\}$, lexical space {T,F,1,0} and lexical-to-value mapping {T$\mapsto$ $true$, F$\mapsto$ $false$, 1$\mapsto$ $true$, 0$\mapsto$ $false$}. Data values can be represented by typed literals or plain literals, where *typed literals* are combinations of string and datatype URIs, while *plain literals* are simply strings, with optional language tag. E.g., "1"^^xsd:boolean is a typed literal, while "1" is a plain literal. Therefore, when we extend OWL datatyping to support predicates, we should not simply replace datatypes with predicates,

but let them co-exist in a proper framework.

Secondly, an OWL datatype interpretation is relativised to a datatype map, which is a partial mapping from datatype URIs to datatypes; e.g., $\mathbf{M}_{d1} = \{\langle \text{xsd:string}, \mathit{string}\rangle, \langle \text{xsd:integer}, \mathit{integer}\rangle\}$. *Unsupported datatypes*, which are not included in a given datatype map, are interpreted as any subsets of the datatype domain. Therefore, the datatype domain (of a datatype interpretation) is expected to be unfixed, which is different from the (datatype) domain in existing approaches.

Thirdly, OWL advocates a more user-friendly style of datatyping than what the existing approaches provide. OWL provides a kind of datatype expressions, called *enumerated datatypes*, of the form oneOf($l_1, \ldots, l_n$), where $l_1, \ldots, l_n$ are literals, which is interpreted as the union of all the interpretation of $l_i$ ($1 \leq i \leq n$). It is expected that it supports more expressive datatype expressions, to represent user-defined datatypes and user-defined predicates. Furthermore, it is desirable that the interpretation of negated predicate is relativised to the value space of the related datatypes; e.g., $\overline{>5}$ is interpreted as $V(\mathit{integer})\backslash >_5^{\mathbf{D}}$ but not $\Delta_{\mathbf{D}}\backslash >_5^{\mathbf{D}}$. Therefore, the interpretation of $\overline{>5}$ will not be affected by the existence of other datatypes in a datatype map.

We extend OWL datatyping with datatype predicates by a revised definition of datatype groups, which was first presented in [5] and was meant to be an extension of DAML+OIL datatyping. Unlike the original definition, the revised definition of datatype groups is completely compatible with OWL datatyping. We show that the predicate conjunctions over datatype groups can be easily reduced to those over concrete domains. We then propose OWL-E, a language extending OWL DL with datatype expression axioms, as well as the datatype group-based class constructors to allow the use of datatype expressions in class restrictions. The novelty of OWL-E is that it enhances OWL DL with much more datatype expressiveness and it is still decidable. Of course, we will need a full paper to present details of the above.

## References


[1] Franz Baader and Philipp 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.

[2] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein eds. OWL Web Ontology Language Reference. URL `http://www.w3.org/TR/owl-ref/`, Feb 2004.

[3] Ian Horrocks and Ulrike 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.

[4] C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

[5] Jeff Z. Pan and Ian Horrocks. Web Ontology Reasoning with Datatype Groups. In *Proc. of the 2nd International Semantic Web Conference (ISWC2003)*, 2003.

# Using Non-Primitive Concept Definitions for Improving DL-based Knowledge Bases

Ronald Cornet, Ameen Abu-Hanna
Department of Medical Informatics
Academic Medical Centre, Amsterdam, The Netherlands
{r.cornet,a.abu-hanna}@amc.uva.nl

April 29, 2004

**Abstract**

Medical Terminological Knowledge Bases contain a large number of primitive concept definitions. This is due to the large number of natural kinds that are represented, and due to the limits of expressiveness of the Description Logic used. The utility of classification is reduced by these primitive definitions, hindering the knowledge modeling process. To better exploit the classification utility, we devise a method in which definitions are assumed to be non-primitive in the modeling process. This method aims at the detection of: duplicate concept definitions, underspecification, and actual limits of a DL-based representation. This provides the following advantages: duplicate definitions can be found, the limits of expressiveness of the logic can be made more clearly, and tacit knowledge is identified which can be expressed by defining additional concept properties. Two case studies demonstrate the feasibility of this approach.

## 1 Introduction

Medical terminological knowledge bases (TKBs) represent knowledge about concepts, relationships and terms, in the domain of medicine. For example, a concept may be defined as "inflammation of the membranes of the brain or spinal cord", and described by the synonymous terms "cerebrospinal meningitis" and "meningitis". TKBs provide an invaluable source of structured medical knowledge, serving a range of purposes.

Advantages of representing this knowledge using Description Logics (DL) include the explicit semantics of the represented knowledge and the possibility to perform automatic reasoning based on this knowledge. The prominent reasoning tasks are satisfiability and subsumption. To infer subsumption, concept definitions with necessary and sufficient conditions are required. We will refer to such definitions as non-primitive definitions (sometimes referred to by others as "equalities"), whereas a primitive concept definition (sometimes referred to by others as "inclusion") specifies only necessary conditions. It is however in general not possible to define all concepts in a non-primitive manner. This is well described in [5]: "There are a large number of concepts that are unclassifiable by virtue of being natural kinds. The problem

is exacerbated by a large number of "fake" primitives, concepts which are primitive only because their definitions cannot be expressed in the restricted language. Since these reduce the utility of classification, using classification's efficiency as the design criterion misplaces emphasis."

Hence, there are various impediments to fully exploit the reasoning strengths that Description Logics offer. This means that in practice, classification may be overlooked that could have been inferred if concept definitions would have been non-primitive. This paper describes the possibilities of using non-primitive concept definitions in the process of knowledge modeling. We describe a method, developed to increase the utility of classification, especially during the knowledge modeling process. This method aims at utilizing DL inference services for the detection of: duplicate concept definitions, underspecification, and actual limits of a DL-based representation. Duplicate definitions should generally not occur in a knowledge base, and underspecification may point at tacit (i.e. not represented by a concept definition) knowledge. Minimizing tacit knowledge will increase the possibilities for distinguishing concepts based on their definitions, and may improve the model by reducing the number of primitive definitions.

We describe the knowledge modeling process in Section 2 and then explain our method in detail in Section 3. Results of the application of the methods in a case study are presented in Section 4 and discussed in section 5. Section 6 concludes this paper.

## 2 The Knowledge Modeling Process

Medical TKBs have grown in size and complexity. This growth has been stimulated by the availability of computers and the potential of using medical TKBs for a wide range of purposes. The complexity has increased due to the possibility of using representation formalisms that allow for more elaborate specification of concept definitions. Medical TKBs evolved from simple taxonomies to semantic networks with (informal and formal) concept definitions. An example of such a system is SNOMED CT[1], a terminological system consisting of approximately 350,000 concepts. Maintenance of systems this large needs to be supported as much as possible in order to reduce modeling errors. To this end we have started a project aiming at assessing and improving the quality of Medical TKBs. Previously, we have discussed the possibilities of detecting inconsistencies in concept definitions [3], and others have focused on this issue as well (e.g. [7]). However, (logical) inconsistencies are not the only modeling error that can occur. Generally, a good terminological system should fulfill a number of desiderata [1]. A formal, concept-oriented approach to modeling terminological knowledge can largely contribute to fulfill a number of these desiderata, for example providing formal definitions and multiple consistent views. However, representation alone is not sufficient. To fully exploit the advantages of formally represented knowledge, services, such as inference services, are indispensible. Standard Description Logic inference services (satisfiability and subsumption testing) provide a solid basis for supporting

[1] `http://www.snomed.org/`

1. InfectiousDisease $\equiv$ Disease $\sqcap \exists$ involves Infection
2. LiverDisease $\equiv$ Disease $\sqcap \exists$ location Liver
3. ViralHepatitis $\equiv$ InfectiousDisease $\sqcap \exists$ location Liver $\sqcap \exists$ cause Virus
4. DuplicateViralHepatitis $\equiv$ LiverDisease $\sqcap \exists$ involves Infection $\sqcap \exists$ cause Virus
5. PrimViralHepatitis $\sqsubseteq$ InfectiousDisease $\sqcap \exists$ location Liver
6. PrimDuplicateViralHepatitis $\sqsubseteq$ LiverDisease $\sqcap \exists$ involves Infection
7. ViralHepatitisTypeA $\equiv$ InfectiousDisease $\sqcap \exists$ cause HepatitisAVirus $\sqcap \exists$ location Liver

Figure 1: Examples of primitive and non-primitive concept definitions

knowledge modeling, but do not support all of the modeling process. For example, these services do not by default contribute to ensuring definition of non-redundant, unambiguous definitions that explicitly and maximally capture the semantics. However, although this is not supported directly, it is possible to utilize DL inference services for tasks such as ensuring definition of non-redundant, unambiguous definitions.

In the next section we focus on methods that utilize DL inference services for detection of concepts that are inadvertently defined more than once, and concepts that may not be exhaustively defined. With exhaustive definitions we mean definitions that maximally capture the semantics of the defined concept. Duplicate definitions should be prevented, as they introduce redundancy into the knowledge base. The motivation for focusing on detection (and reduction) of non-exhaustive definitions, is that non-exhaustively defined concepts may point at tacit (i.e. not represented by a concept definition) knowledge. Minimizing tacit knowledge will increase the possibilities for distinguishing concepts based on their definitions, and may improve the model by reducing the number of primitive definitions, supporting among others more elaborate classification.

## 3 Use of non-primitive Definitions

An essential feature of Description Logics is the distinction between definitions that state only necessary conditions (which we refer to as primitive definitions) and definitions that state necessary and sufficient conditions (non-primitive definitions). Non-primitive definitions facilitate the inference of classification (subsumption) of concepts. These definitions also make it possible to detect equivalent concepts (which actually boils down to mutual subsumption). For example, given the definitions 1 and 2 from Figure 1, equivalence of the concepts ViralHepatitis and DuplicateViralHepatitis, defined in 3 and 4, can be inferred. However, if these concept definitions would have been primitive, as in definitions 5 and 6, equivalence would not have been detected, and duplicate concepts would remain undetected. Furthermore, the concepts defined in 5 and 6 contain tacit knowledge, as these definitions do not state the viral cause of the disease.

As the domain of medicine consists of many natural kinds, for which no necessary and sufficient conditions exist, many disease concepts in Medical TKBs can only be

defined in a primitive manner. As a result of this, much of the inferential potential is lost, as the example above demonstrates. Another example would be missed classification. For example, given definition 7, ViralHepatitisTypeA would be rightly classified as a ViralHepatitis, but not as a PrimViralHepatitis, although it should be.

Although it is inevitable to have many primitive definitions in a Medical TKB, it makes sense to use the inferential powers of DL reasoners in the modeling process by stating, as an assumption, non-primitivity of all relevant concept definitions.

The first step is to determine which concepts in a knowledge base might have duplicate definitions or contain tacit knowledge that needs to be made explicit. Generally, medical TKBs consist of various "modules" that are used to define the concepts in the primary category of interest. Such TKBs can be regarded as a collection of subtrees, where the roots of these subtrees can be for example Diseases, Anatomical Components and Micro-Organisms. The Micro-Organisms subtree is used in the definitions of etiology of diseases, as is also shown in definition 7 in Figure 1. As a TKB is generally focused on one subtree, in this case diseases, it may be expected that the other modules are far from exhaustively defined. This will result in many equivalent concepts, not only in the respective subtrees, but also in the Diseases subtree, as concepts in this subtree are defined using concepts that were considered equivalent. There are two options in such a situation. The first option is to initially focus on the respective subtrees (such as Micro-Organisms), in order to find duplicates and make tacit knowledge explicit. The second option is to leave these subtrees out of account (i.e. treat the concepts as base symbols), and focus on the subtree of primary interest, which contains disease concepts in this example.

The next step is to find all concepts that have a definition of the form: $B \sqsubseteq A$, where A is a concept name. There is no use in changing these definitions to non-primitive definitions, as this will provide a trivial equivalence ($B \equiv A$). These concepts may be expected not to represent duplicate definitions (as they have been explicitly defined), but B either represents a natural kind or is underspecified. As concepts of this form are easily recognizable, they can be studied separately. The last step is to redefine all relevant concepts (e.g. disease concepts) that are not of the form $B \sqsubseteq A$ as non-primitive.

When the TKB has been altered according to the steps mentioned above, it can be classified with a DL reasoner. This classification will result in clusters of equivalent concepts. These clusters then have to be analyzed by hand. This analysis will provide two types of outcomes. First, a set of concepts that form duplicate definitions, which were previously undetected due to their primitive definitions. Second, a set of concepts for which the differences among them do exist, but are not represented. In the latter case, which we will refer to as underspecification, the knowledge base can potentially be improved by making explicit the implicit knowledge that distinguishes one concept from another. If this distinction can not be made explicit, this can either be caused by the lack of characteristic features of the concept (i.e. it is a natural kind), or by limitations of the used DL. We have previously performed a study on the use of terms indicating the need for certain constructors [2]. The above-mentioned method will contribute to gaining insight into the need for specific constructors by making the needs more precise and related to a specific TKB.

Cachexia ≡ Medical_Diagnosis ⊓ ∃ involved_system Metabolic_system
Starvation ≡ Medical_Diagnosis ⊓ ∃ involved_system Metabolic_system
Hypermagnesaemia ≡ Metabolic_Disorder ⊓ ∃ involved_component Body_fluids
Hypomagnesaemia ≡ Metabolic_Disorder ⊓ ∃ involved_component Body_fluids
Hypercalcaemia ≡ Metabolic_Disorder ⊓ ∃ involved_component Body_fluids
Hypocalcaemia ≡ Metabolic_Disorder ⊓ ∃ involved_component Body_fluids

Figure 2: Examples of equivalent concept definitions. Six concepts are defined, but classification results in two different concepts: Cachexia and Starvation as one concept, and Hypermagnesaemia, Hypomagnesaemia, Hypercalcaemia, and Hypocalcaemia as another concept

# 4 Results of two Case Studies

The approach is applied to a medical TKB on Reasons for Admission in Intensive care (DICE) [4] and on the Foundational Model of Anatomy (FMA)[2]. The DICE knowledge base, which is under development at the institution of the authors, contains about 2500 concepts, of which 1456 Reasons for Admission. Reasons for Admission comprise both diseases and procedures that require intensive care and monitoring of patients. We have applied the above-mentioned methods to detect duplicate definitions in the system, and to determine possibilities to improve modeling by reducing underspecifications. As we are aware of underspecification in domains other than Reasons for Admission (such as anatomy, and etiology), we have limited our evaluation to the Reasons for Admission taxonomy.

The case study on FMA has been performed mainly to determine to what extent the findings in DICE were knowledge-base-specific. FMA, developed by the University of Washington, provides about 69000 concept definitions, describing anatomical structures, shapes, and other entities, such as coordinates (left, right, etc.).

## 4.1 Results of the Case Study on DICE

The DICE knowledge base, which was implemented as a simple TBox with an empty ABox, was represented using Knowledge Representation System Specification (KRSS) syntax [6]. This made it straightforward to discern primitive definitions from non-primitive ones, simply by performing a text-based search in the KRSS file. Replacement of primitive definitions by non-primitive definitions for appropriate concepts out of 1456 Reasons for Admission, resulted in 108 (7%) concept definitions that were primitive, and 1348 (93%) concept definitions that were non-primitive. As was explained in Section 3, all primitive definitions were of the form B ⊑ A, for example: Eclampsia ⊑ Hypertension_induced_by_Pregnancy.

RACER[3] was used to classify the resulting TBox. This resulted in 24 unsatisfiable concepts, which we will discuss further in Section 5. Of the remaining 1432 satisfiable concept names (in the Reason for Admission module), 1160 (81%) had a unique defini-

[2] http://sig.biostr.washington.edu/projects/fm/
[3] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

Table 1: Results of detection of equivalently defined concepts in the Reason of Admission module of DICE. The first column shows the size of clusters, the second column the number of clusters with the specified size, the last column shows the total number of concepts in the clusters of the specified size (i.e. the product of cluster size and number of clusters).

| **# equivalent definitions** | **# clusters** | **# concepts in clusters** |
|---|---|---|
| 2 | 60 | 120 |
| 3 | 23 | 69 |
| 4 | 8 | 32 |
| 6 | 2 | 12 |
| 7 | 4 | 28 |
| 11 | 1 | 11 |
| | 98 | 272 |

tion, and 272 concepts (19%) were equivalent to one or more other concepts. As shown in Figure 2, classification will render multiple concepts with equivalent definitions as one concept. The 272 concepts could be traced back to 98 definitions that were used twice or more, as is shown in Table 1. There were 60 tuples of equivalent definitions (such as Cachexia and Starvation), and 1 cluster with 11 concepts. This last cluster contained concepts as diverse as Water_Depletion and Familial_Periodic_Paralysis.

### 4.2 Result of the Case Study on FMA

The FMA knowledge base, which is implemented as a frame-based model in Protege[4], has been migrated to DL, where specified slot-fillers in the frame-based representation were interpreted as existentially quantified roles. This simple TBox, with an empty ABox, was represented using KRSS syntax. Replacement of primitive definitions by non-primitive definitions for appropriate concepts, resulted in a DL-based representation of all of FMA that contained about 50% primitive and 50% non-primitive concept definitions. We were not able to classify the full TBox with RACER, probably because of the use of roles and their inverses (e.g. part_of and part), leading to cyclic definitions. Because of this, we limited the case study to "Organs". Of the 3826 concept definitions, 2659 (69%) were non-primitive, and 1167 (31%) were primitive. Classification with RACER resulted in 3323 concepts (87%) that had a unique definition, and 503 concepts (13%) that were equivalent to one or more other concepts. These 503 concepts could be traced back to 160 definitions that were used twice or more, as is shown in Table 2. There were 106 tuples of equivalent definitions, and 1 cluster with 54 equivalent concepts. This cluster contained a variety of ligaments of joints, such as Interosseous_ligament_of_carpometacarpal_joint and Palmar_ligament_of_left_fifth_carpometacarpal_joint.

[4] `http://protege.stanford.edu/`

Table 2: Results of detection of equivalently defined concepts in the Organ module of FMA. The first column shows the size of clusters, the second column the number of clusters with the specified size, the last column shows the total number of concepts in the clusters of the specified size (i.e. the product of cluster size and number of clusters).

| # equivalent definitions | # clusters | # concepts in clusters |
|---|---|---|
| 2 | 106 | 212 |
| 3 | 33 | 99 |
| 4 | 10 | 40 |
| 5 | 4 | 20 |
| 6 | 3 | 18 |
| 7 | 1 | 7 |
| $\geq$8 | 3 | 107 |
| | 160 | 503 |

### 4.3 Explanation of equivalence

As described in Section 3, there can be various explanations for concept equivalence. Concepts can be actually duplicately defined, but can also be underspecified. This underspecification is inevitable when concepts are natural kinds, or when concept properties can not be expressed due to limits of the used DL. Avoidable underspecification indicates tacit knowledge, that could be made explicit by making definitions more exhaustive. Although a full evaluation of equivalent definitions has to be performed, a first study provides markable results.

In DICE, only 4 tuples of concept definitions were found that are potential duplicates, but this needs to be discussed with domain experts. Examples of such concepts are "reconstruction of artery" versus "arterial angioplasty" and "biliary drainage" versus "drainage of biliary duct".

Apart from these potentially true duplicates, all equivalent concepts differ in meaning in a way that is not represented in the knowledge base. In DICE, a small number of natural kinds was found, which were to a large extent syndromes and/or eponyms. Examples of these are "Adult Respiratory Distress Syndrome", "Wilms' tumour", and "Wolff-Parkinson-White syndrome".

Both DICE and FMA originally have a frame-based representation, and both have been migrated to DL in order to be able to perform the experiments described. In DICE, a small number of concepts was found that explicitly mentioned negation, which can not be represented using frames. Examples of these are "bleeding" versus "non-bleeding" and "obstructive" versus "non-obstructive". This difference could be explicitly represented using a DL that allows for negation.

The vast majority of concepts that were defined as primitive or that were non-uniquely defined, demonstrated underspecification that seemed to be relatively easy to avoid. This means that it is possible and appropriate to make definitions more

exhaustive by adding conditions.

Possible improvements to DICE can be determined by studying equivalent concepts. For example, equivalence of hypocalcaemia and hypercalcaemia can be resolved by making explicit the level involved: "below normal" resp "above normal". Equivalence of hypercalcaemia and hypermagnesaemia is explained by the lack of specification of the involved chemical elements (calcium resp magnesium). These examples demonstrate required extensions to the knowledge base, as chemical elements and levels are currently not defined in the knowledge base. There were however also many concepts that can be refined using concept and roles that are already available in the knowledge base. For example "Meningococcal meningitis" was defined as a "Bacterial meningitis", without mention of a relation with a concept "Meningococcus" through an "etiology" role. Hence, making such a definition more exhaustive is not only straightforward, it is even required, if one wants to classify meningococcal meningitis as a disease that is caused by meningococcus. Making concept definitions more exhaustive using readily available concepts and roles, also seemed possible in FMA. For example, "Synovial tendon sheath of flexor hallucis longus" and "Synovial tendon sheath of tibialis anterior", can be distinguished by explicitly relating them to "flexor hallucis longus", and "tibialis anterior", respectively.

# 5 Discussions

The two case studies demonstrate the feasibility and usability of our approach. In order to assess the overall applicability of the approach, it is useful to look further into the peculiarities of the knowledge bases used in the case studies. We will discuss these below. Thereafter, we will shortly discuss an alternative approach that could be used instead of our method, namely structural subsumption.

## 5.1 Modeling Issues

The knowledge bases that have been studied exhibit a number of properties that render them suitable for the method described in this paper. Both DICE and FMA are represented as simple TBoxes. This implies that no atomic concept occurs more than once as left-hand side, and the left-hand side of all axioms are atomic concepts (so no arbitrary concept expressions are allowed on the left-hand side). Moreover, the DICE TBox is acyclic, meaning that no concept name is defined with reference to itself (such as for example: Human $\equiv$ Animal $\sqcap \forall$ hasParent Human). The FMA TBox contains cycles, caused by the use of roles and their inverses (e.g. part_of and part). The similarity and simplicity of both knowledge bases can be explained by the fact that the DL-based representations are the result of a migration process from a frame-based representation, as described in [3].

The FMA TBox was coherent, and the DICE TBox contained only a small number of unsatisfiable concepts (due to the migration process). Having a minimum of unsatisfiable concepts is important as unsatifiability "propagates" over existentially quantified roles. Assume that B is an unsatisfiable concept, then C, defined as C $\sqsubseteq$ A $\sqcap \exists$ R B, will also be unsatisfiable.

In order to minimize the number of unsatisfiable concepts, no disjointness between concepts was explicitly stated. This can be explained by the following example. Suppose the original knowledge base contains two primitive definitions: $C_1 \sqsubseteq A \sqcap \exists R\ B$, and $C_2 \sqsubseteq A \sqcap \exists R\ B$, and $C_1$ and $C_2$ are stated to be disjoint. Applying our method would change these definitions to non-primitive definitions, which would render $C_1$ and $C_2$ equivalent. But as they are also disjoint, each would be inferred to be unsatisfiable.

The DICE TBox is defined using the language $\mathcal{ALCQ}$, the FMA TBox can be expressed with $\mathcal{ALCI}$. This means that for example role hierarchies and transitive roles are not used in these TKBs. Actually, DICE was modeled using Structure-Entity-Part (SEP) triplets, described in [8], in order to prevent the use of transitive roles and role hierarchies.

It needs to be determined whether the method is also useful for more complex knowledge bases. Issues that increase the complexity of knowledge bases are the use of a more expressive language, cyclic definitions, use of concept inclusion axioms with concept expressions on the left-hand side (instead of only atomic concepts), and allowing multiple definitions of a concept.

### 5.2 Alternative Approach: Structural Subsumption

There are two reasons for discussing structural subsumption as an alternative approach. The first reason is the fact that the case studies involved relatively simple knowledge bases. The second reason was that a superficial inspection of equivalent definitions of concepts indicated that most of them were not only logically equivalent (as definitions 3 and 4 from Figure 1), but even structurally equivalent, as shown in Figure 2. Structural subsumption could prove useful for knowledge bases for which the computational cost of classification is too high, such as for example the complete FMA. The advantage of using a structural subsumption algorithm is that it is generally cheaper in terms of computational costs, but it has the drawback that it is not complete.

## 6 Conclusions

We have applied the inferential powers of DL reasoners to detect concepts that are equivalently defined within a knowledge base. In order to be able to find such concepts, we have considered relevant concept definitions as non-primitive. This results in clusters of concepts which have equivalent definitions.

Two case studies show that the size of such clusters varied mainly from 2 to 7. For the vast majority of concept definitions that turned out to be equivalent it is possible to make them more exhaustive by adding conditions that distinguish between them. For a minority of the equivalent concepts there seemed to be no possibilities of making the definition more exhaustive, as these concepts represented natural kinds, or could not be defined due to limits of the underlying representation. The case study on DICE revealed only a few duplicate definitions in the knowledge base.

Overall, it can be concluded that applying the methods described in this paper, contributes to gaining insight in tacit knowledge, which is unrepresented in a knowledge base. Making this knowledge explicit by means of refining concept definitions improves the knowledge base, and results in more exhaustive concept definitions. However, it can not be guaranteed that these more exhaustive definitions will now provide both necessary and sufficient conditions. Therefore, it needs to be determined whether this method can result in an actual decrease in the number of primitive concept definitions in knowledge bases, which would increase the powers of inference based on the knowledge base. The successful application of our method to two knowledge bases in the field of medicine, makes it likely to be applicable to other domains as well.

## References


[1] J. J. Cimino. Desiderata for controlled medical vocabularies in the twenty-first century. *Methods of Information in Medicine*, 37(4-5):394–403, 1998.

[2] R. Cornet and A. Abu-Hanna. Usability of expressive description logics – a case study in UMLS. *Proc AMIA Symp*, pages 180–4, 2002.

[3] R. Cornet and A Abu-Hanna. Using description logics for managing medical terminologies. In M. Dojat, E. Keravnou, and P. Barahona, editors, *9th Conference on Artificial Intelligence in Medicine in Europe, AIME*, pages 61–70, Protaras, Cyprus, 2003. Springer.

[4] N. F. de Keizer, A. Abu-Hanna, R. Cornet, J. H. Zwetsloot-Schonk, and C. P. Stoutenbeek. Analysis and design of an ontology for intensive care diagnoses. *Methods of Information in Medicine*, 38(2):102–12, 1999.

[5] Jon Doyle and Ramesh Patil. Two theses of knowledge representation: Language restrictions, taxonomic classifications, and the utility of representation services. *Artificial Intelligence*, 48(3):261–298, 1991.

[6] PF Patel-Schneider and B Swartout. Description-logic knowledge representation system specification from the krss group of the arpa knowledge sharing effort. Technical report, KRSS Group of the ARPA Knowledge Sharing Effort, 1 november 1993 1993.

[7] E. B. Schulz, J. W. Barrett, and C. Price. Semantic quality through semantic definition: refining the read codes through internal consistency. *Proc AMIA Annu Fall Symp*, pages 615–9, 1997.

[8] S. Schulz, M. Romacker, and U. Hahn. Part-whole reasoning in medical ontologies revisited–introducing sep triplets into classification-based description logics. *Proc AMIA Symp*, pages 830–4, 1998.

# Extended Query Facilities for Racer and an Application to Software-Engineering Problems

Volker Haarslev[□], Ralf Möller[◇],
Ragnhild Van Der Straeten[△] and Michael Wessel[○]

Concordia University[□]
Montreal, Canada
haarslev@cs.concordia.ca

Technical University of Hamburg-Harburg[◇]
Hamburg-Harburg, Germany
r.f.moeller@tuhh.de

Vrije Universiteit Brussel[△]
Brussels, Belgium
rvdstrae@vub.ac.be

University of Hamburg[○]
Hamburg, Germany
mwessel@informatik.uni-hamburg.de

**Abstract**

This paper reports on a pragmatic query language for Racer. The abstract syntax and semantics of this query language is defined. Next, the practical relevance of this query language is shown, applying the query answering algorithms to the problem of consistency maintenance between object-oriented design models.

## 1 Motivation

Practical description logic (DL) systems such as Racer [3] offer a functional API for querying a knowledge base (i.e., a tuple of T-box and A-box). For instance, Racer provides a query function for retrieving all individuals mentioned in an A-box that are instances of a given query concept. Let us consider the following A-box: $\{has\_child(alice, betty), has\_child(alice, charles)\}$. If we are interested in finding individuals for which it can be proven that a child exists, in the Racer system, the function *concept_instances* can be used. However, if we would like to find all tuples of individuals $x$ and $y$ such that a common parent exists, currently, it is not possible to express this in sound and complete DL systems such as, for instance, Racer. Other logic-based representation systems, such as, e.g., LOOM [6], however, have offered query languages suitable for expressing the second query right from the beginning. In this paper we define syntax and semantics of a query language similar to that of LOOM. Users of description logic systems such as Racer already demonstrated the demand of such a query language for sound and complete DL systems [9], and this paper evaluates the practical relevance of the current implementation for query answering algorithms in Racer-1-7-19 using an application to software engineering problems.

## 2 The New Racer Query Language - nRQL

In the following we describe the *n*ew *R*acer *Q*uery *L*anguage, also called *nRQL* (pronounce: Nercle). We start with some auxiliary definitions:

**Definition 1 (Individuals, Variables, Objects)** Let $\mathcal{I}$ and $\mathcal{V}$ be two disjoint sets of *individual names* and *variable names*, respectively. The set $\mathcal{O} =_{def} \mathcal{V} \cup \mathcal{I}$ is the set of *object names*. We denote variable names (or simply variables) with letters $x, y, \ldots$; individuals are named $i, j, \ldots$; and object names $a, b, \ldots$. ■

*Query atoms* are the basic syntax expressions of nRQL:

**Definition 2 (Query Atoms)** Let $a, b \in \mathcal{O}$; $C$ be an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ [4] concept expression, $R$ a role expression, $P$ one of the concrete domain predicates offered by Racer; $f = f_1 \circ \cdots \circ f_n$ and $g = g_1 \circ \cdots \circ g_m$ be feature chains such that $f_n$ and $g_m$ are attributes (whose range is defined to be one of the available *concrete domains* offered by Racer, or $f, g$ are individuals from one of the offered concrete domains which means that $m, n = 1$ and $f_1, g_1$ are 0-ary attributes). Then, the list of *nRQL atoms* is given as follows:

- Unary concept query atoms: $C(a)$
- Binary role query atoms: $R(a, b)$
- Binary constraint query atoms: $P(f(a), g(b))$
- Unary bind-individual atoms: $bind\_individual(i)$
- Unary has-known-successor atoms: $has\_known\_successor(a, R)$
- Negated atoms: If $rqa$ is a nRQL atom, then so is $\backslash(rqa)$, a so-called *negation as failure atom* or simply *negated atom.* ■

We give some examples of the various atoms and assume throughout the paper that $betty \in \mathcal{I}$. Note that $(woman \sqcap (\neg mother))(betty)$ and $woman(x)$ are unary concept query atoms; $has\_child(x, y)$ and $has\_child(betty, y)$ are binary role query atoms; $\texttt{string=}(has\_father \circ has\_name(x), has\_name(y))$ as well as $\geq(has\_age(x), 19)$ are examples of binary constraint query atoms. Finally, $\backslash(woman(x))$ is an example of a negated atom. The rationale for introducing unary bind-individual and has-known-successor atoms will become clear later. Both are related to effects caused by negated atoms. We can now define nRQL queries:

**Definition 3 (nRQL Query Bodies, Queries & Answer Sets)** A *nRQL Query* has a *head* and a *body*. *Query bodies* are defined inductively as follows:

- Each nRQL atom $rqa$ is a body; and
- If $b_1 \ldots b_n$ are bodies, then the following are also bodies:
  - $b_1 \wedge \cdots \wedge b_n$, $b_1 \vee \cdots \vee b_n$, $\backslash(b_i)$

We use the syntax $body(a_1, \ldots, a_n)$ to indicate that $a_1, \ldots, a_n$ are all the objects ($a_i \in \mathcal{O}$) mentioned in *body*. A *nRQL Query* is then an expression of the form

$$ans(a_{i_1}, \ldots, a_{i_m}) \leftarrow body(a_1, \ldots, a_n),$$

$ans(a_{i_1}, \ldots, a_{i_m})$ is also called the *head*, and $(i_1, \ldots, i_m)$ is an index vector with $i_j \in 1 \ldots n$. A *conjunctive nRQL query* is a query which does not contain any $\vee$ and $\setminus$ operators. ■

Before we consider atoms with variables, we define truth of *ground query atoms.* A ground query atom does not reference any variables. To define truth of ground query atoms, we will need the standard notion of *logical implication* or *logical entailment.* We first start with *positive atoms* – atoms which are not negated:

**Definition 4 (Entailment of Positive Ground Query Atoms)** Let $\mathcal{K}$ be an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ knowledge base. A knowledge base, KB for short, is simply a T-box/A-box tuple: $\mathcal{K} = (\mathcal{T}, \mathcal{A})$.

A *positive ground query atom rqa (i.e., rqa doesn't reference variables and is not negated)* is logically entailed (or implied) by $\mathcal{K}$ iff every model $\mathcal{I}$ of $\mathcal{K}$ is also a model of $rqa$. In this case we write $\mathcal{K} \models rqa$. Moreover, if $\mathcal{I}$ is a model of $\mathcal{K}$ ($rqa$) we write $\mathcal{I} \models \mathcal{K}$ ($\mathcal{I} \models rqa$).

We therefore have to specify when $\mathcal{I} \models rqa$ holds. In the following, if $rqa$ references individuals $i, j$, it will always be the case that $i, j \in \mathsf{inds}(\mathcal{A})$. From this it follows that $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and $j^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for any $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\mathcal{I} \models \mathcal{K}$:

- If $rqa = C(i)$, then $\mathcal{I} \models rqa$ iff $i^{\mathcal{I}} \in C^{\mathcal{I}}$.
- If $rqa = R(i, j)$, then $\mathcal{I} \models rqa$ iff $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$.
- If $rqa = P(f(i), g(j))$, then $\mathcal{I} \models rqa$ iff
  - $c_i = f^{\mathcal{I}}(i^{\mathcal{I}})$,
  - $c_j = g^{\mathcal{I}}(j^{\mathcal{I}})$,
  - $(c_i, c_j) \in P^{\mathcal{I}}$; moreover,
  - if $f = f_1 \circ \cdots \circ f_n$, then we require that for all $m \in 1 \ldots n-1$ with $k = (f_1 \circ \cdots \circ f_m)^{\mathcal{I}}(i^{\mathcal{I}})$ there is some $j \in \mathsf{inds}(\mathcal{A})$ such that $j^{\mathcal{I}} = k$; and analogously for $g$.
- If $rqa = (i = j)$, then $\mathcal{I} \models (i = j)$ iff $i^{\mathcal{I}} = j^{\mathcal{I}}$.
- If $rqa = has\_known\_successor(i, R)$, then $\mathcal{I} \models rqa$ iff for some $j \in \mathsf{inds}(\mathcal{A})$: $\mathcal{I} \models R(i, j)$. ■

It is important to note that the properties of roles and concepts referenced in the query atoms are defined in the knowledge base $\mathcal{K}$. For example, if the role *has_descendant* has been declared as transitive in $\mathcal{K}$, then *has_descendant* will be transitive in the queries as well, since in models of $\mathcal{K}$ $has\_descendant^{\mathcal{I}} = (has\_descendant^{\mathcal{I}})^+$ must hold. If $has\_father$ is declared as a feature, then it will behave as a feature in the queries as well. Also note that, according to Definition 2, atoms of the form $rqa = (i = j)$ are not really query atoms. However, these atoms are used to *replace bind_individual* atoms, see below.

The complex semantic condition enforced on the binary constraint query atoms such as $rqa = P(f(i), g(j))$ with $f = f_1 \circ \cdots \circ f_n$ and $g = g_1 \circ \cdots \circ g_m$ makes it possible

to substitute such an atoms with the conjunction $f_1(i, i_1) \wedge \cdots \wedge f_{n-1}(i_{n-2}, i_{n-1}) \wedge g_1(j, j_1) \wedge \cdots \wedge g_{m-1}(j_{m-2}, j_{m-1}) \wedge P(f_n(i_{n-1}), g_m(j_{m-1}))$.

Now that we have defined truth of of positive ground query atoms, we can define truth of arbitrary ground query atoms:

**Definition 5 (Truth of Ground Query Atoms)** Let $rqa$ be a ground query atom. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base (T-box/A-box tuple). A ground atom $rqa$ is either TRUE in $\mathcal{K}$ (we write $\mathcal{K} \models_{NF} rqa$) or FALSE in $\mathcal{K}$ (we write $\mathcal{K} \not\models_{NF}$). The relationship $\models_{NF}$ resp. trueness of ground query atoms is inductively defined as follows:

- If $rqa$ is positive (does not contain "\"): $\mathcal{K} \models_{NF} rqa$ iff $\mathcal{K} \models rqa$
- Otherwise: $\mathcal{K} \models_{NF} \backslash(rqa)$ iff $\mathcal{K} \not\models_{NF} rqa$ ■

It is important to note that for each query body or atom $q$, $q$ is TRUE iff $\backslash(q)$ is FALSE, and vice versa. Note that this does not hold for the usual entailment relationship. For example, consider the A-box $\{woman(betty)\}$. Given $\mathcal{K} =_{def} (\emptyset, \mathcal{A})$, $woman(betty)$ is TRUE, and $mother(betty)$ is FALSE, since we cannot prove that $betty$ is a mother. Thus, $\backslash(mother(betty))$ is TRUE. In contrast, $\neg mother(betty)$ is obviously FALSE. Moreover, $(mother \sqcup \neg mother)(betty) = \top(betty)$ is *not* the same as $(mother(betty)) \vee (\neg mother(betty))$.

In order to check whether $\mathcal{K} \models_{NF} rqa$, we can use the basic consistency checking and A-box retrieval methods offered by Racer. The symbol "$\models_{NF}$" shall remind the reader of the employed "Negation as Failure" semantics (i.e., suppose $rqa$ is positive, then $\mathcal{K} \models_{NF} \backslash(rqa)$ iff $\mathcal{K} \not\models rqa$, which means $\backslash(rqa)$ is TRUE in $\mathcal{K}$, see below for examples).

The truth definition of ground atoms can be extended to complex *ground query bodies* in the obvious way (i.e., $\mathcal{K} \models_{NF} b_1 \wedge \cdots \wedge b_n$ iff $\forall b_i : \mathcal{K} \models_{NF} b_i$, and analogously for $\vee$ and $\backslash$).

Having defined truth of ground query atoms and bodies, we can specify the semantics of queries which are not ground, but first we need one more piece of notation. The rationale behind the next definition is best understood with an example: consider the query $ans(betty) \leftarrow woman(betty)$. The answer to this query should either be $\emptyset$ (in case $\mathcal{K} \not\models woman(betty)$), or $\{(betty)\}$ (in case $\mathcal{K} \models woman(betty)$). A reasonable statement is that $ans(betty) \leftarrow \backslash(woman(betty))$ should be the complement query of $ans(betty) \leftarrow woman(betty)$. The latter one should therefore return the set $\{\,(i) \mid\ i \in \mathsf{inds}(\mathcal{A})\,\}$, probably without $\{(betty)\}$ (note that $\mathsf{inds}(\mathcal{A})$ returns the set of all individuals mentioned in the A-box $\mathcal{A}$). Thus, within $\backslash(woman(betty))$, $betty$ behaves in fact like a variable. To capture this behavior, we replace the individuals in the atoms with representative variables and use (in)equality statements as follows:

**Definition 6 ($\alpha$-Substitution)** Let $rqa$ be an atom that contains at most one "\" (note that $rqa = \backslash(\backslash(rqa))$). Denote the set of mentioned individuals in $rqa$ as $\mathsf{inds}(rqa)$. Then, $\alpha(rqa)$ is defined as follows:

- If $\mathsf{inds}(rqa) = \emptyset$, then $\alpha(rqa) =_{def} rqa$.

- If $rqa = \mathit{bind\_individual}(i)$, then $\alpha(rqa) =_{def} x_i = i$
- If $rqa = \backslash(\mathit{bind\_individual}(i))$, then $\alpha(rqa) =_{def} x_i \neq i$
- If $rqa$ is not a bind-individual atom, then
  - If $rqa$ is *positive* and $\mathsf{inds}(rqa) = \{i, j\}$ (possibly $i = j$), then $\alpha(rqa) =_{def} rqa_{[i \leftarrow x_i, j \leftarrow x_j]} \wedge (x_i = i) \wedge (x_j = j)$.
  - If $rqa = \backslash(rqa')$ is *negative* and $\mathsf{inds}(rqa) = \{i, j\}$ (possibly $i = j$), then $\alpha(rqa) =_{def} \backslash(rqa'_{[i \leftarrow x_i, j \leftarrow x_j]}) \vee (x_i \neq i) \vee (x_j \neq j)$. ■

Note that $rqa_{[i \leftarrow x_i, j \leftarrow x_j]}$ means "substitute $i$ with $x_i$, and $j$ with $x_j$". For example, $\alpha(R(i,j)) = R(x_i, x_j) \wedge (x_i = i) \wedge (x_j = j)$, but $\alpha(\backslash(R(i,j))) = \backslash(R(x_i, x_j)) \vee (x_i \neq i) \vee (x_j \neq j)$. We extend the definition of $\alpha$ to query bodies in the obvious way. However, we need to bring the bodies into *negation normal form (NNF)* first, such that "\" appears only in front of atoms. This is simply done by applying *DeMorgan's Law* to the query body (from the given semantics it follows that $\backslash(A \wedge B) \equiv \backslash(A) \vee \backslash(B)$, $\backslash(A \vee B) \equiv \backslash(A) \wedge \backslash(B)$, $\backslash(\backslash(A)) \equiv A$). The semantics of a nRQL query can now be paraphrased as follows:

**Definition 7 (Semantics of a Query)** Let $\mathit{ans}(a_{i_1}, \ldots, a_{i_m}) \leftarrow \mathit{body}(a_1, \ldots, a_n)$ be a nRQL query $q$ such that *body* is in NNF. Let $\beta(a_i) =_{def} x_{a_i}$ if $a_i \in \mathcal{I}$, and $a_i$ otherwise; i.e., if $a_i$ is an individual we replace it with its representative unique variable which we denote by $x_{a_i}$. Let $\mathcal{K}$ be the knowledge base to be queried, and $\mathcal{A}$ be its A-box. The *answer set* of the query $q$ is then the following set of tuples:

$$\{\, (j_{i_1}, \ldots, j_{i_m}) \mid \exists j_1, \ldots, j_n \in \mathsf{inds}(\mathcal{A}), \forall m, n, m \neq n : j_m \neq j_n, \\ \mathcal{K} \models_{NF} \alpha(\mathit{body})_{[\beta(a_1) \leftarrow j_1, \ldots, \beta(a_n) \leftarrow j_n]} \,\}$$

Finally, we state that $\{()\} =_{def} \mathsf{TRUE}$ and $\{\} =_{def} \mathsf{FALSE}$. ■

Note that we assume the *unique name assumption (UNA)* for the variables here. However, the implemented query processing engine also offers non-UNA variables (originally meant for breaking up feature chains). For reasons of brevity we decided not to include them in the formal definition here.

Let us briefly discuss some "pathological examples" which explain why we included bind-individual and has-known-successor atoms into nRQL.

Suppose we want to know for which individuals we have explicitly modeled children in the A-box. For this purpose, the query $\mathit{ans}(x) \leftarrow \mathit{has\_know\_successor}(\mathit{has\_child}, x)$ can be used, but also the query $\mathit{ans}(x) \leftarrow \mathit{has\_child}(x, y)$. However, now suppose we want to retrieve the A-box individuals which *do not* have a child. The query $\mathit{ans}(x) \leftarrow \backslash(\mathit{has\_child}(x, y))$ cannot be used, since first the complement of $\mathit{has\_child}(x, y)$ is computed, and then the projection to $x$ is carried out. Thus, $\mathit{ans}(x) \leftarrow \backslash(\mathit{has\_known\_successor}(x, \mathit{has\_child}))$ must be used. Please note that this query is not equivalent to $\mathit{ans}(x) \leftarrow \backslash(\exists \mathit{has\_child}.\top(x))$. To see why, suppose we want to query for *mothers* not having any explicitly modeled children in the A-box. Obviously, these mothers cannot be retrieved with $\mathit{ans}(x) \leftarrow \backslash(\exists \mathit{has\_child}.\top(x))$, since motherhood implies having a child. But this child need not be explicitly modeled in the

A-box. Thus, the query $ans(x) \leftarrow mother(x) \wedge \backslash(has\_known\_successor(x, has\_child))$ must be used. The syntax $ans(x) \leftarrow mother(x) \wedge has\_child(x, \texttt{NIL})$, which we borrowed from the query language of the LOOM system, is also understood by Racer.

We already mentioned that individuals appearing within negated query atoms turn into variables. Suppose $ans(eve) \leftarrow mother(eve)$ returns $\emptyset$. Thus, the query $ans(eve) \leftarrow \backslash(mother(eve))$ will return the complement set w.r.t. all mentioned A-box individuals, e.g. $\{(eve)(doris)(charles)(betty)(alice)\}$. But sometimes, this behavior is unwanted: in this case we can add the additional conjunct $bind\_individual(eve)$. We then get $\{(eve)\}$ for $ans(eve) \leftarrow bind\_individual(eve) \wedge \backslash(mother(eve))$.

# 3 An Example from Software Engineering

In [9], we plead for state-of-the-art DL tools having an extensive query language to be able to maintain consistency between object-oriented design models.

The *de facto* modeling language for the analysis and design of object-oriented software applications is UML [7]. The visual representation of this language consists of several diagram types. Those diagrams represent different views on the system under study. We deliberately confine ourselves to three kinds of UML diagrams: class diagrams representing the static structure of the software application, sequence diagrams representing the behavior of the software application in terms of the collaboration between different objects, and state diagrams modeling the behavior of one single object.

State-of-the-art CASE tools have little support for maintaining the consistency between those different diagrams within the same version of a model or between different versions of a model.

Based on a detailed analysis of all the UML concepts appearing in class, sequence and state diagrams, several consistency conflicts are identified and classified. For an overview of this classification, we refer to [8].

In our approach the relevant subset of the UML metamodel, defining class, state and sequence diagrams, is translated into T-box axioms. As such the different user-defined UML diagrams are translated into A-box assertions. Based on two illustrative consistency conflicts, we argued in [9] that checking for inconsistencies demands an extensive query language. This would allow us to specify UML models and consistency rules in a straightforward and generic way.

The *classless instance* conflict, described in [9] is repeated here and the *infinite containment* conflict is introduced.

## 3.1 Classless instances

The first conflict appears if there are instances in a sequence diagram which do not have any associated class. An example of this conflict is shown in Figure 1, where the object *anATM* is an instance of *ATM* in the sequence diagram on the right side of Figure 1 but this class does not appear in the class diagram on the left side of the same figure.
Classes in a class diagram are represented by the concept *class* in our T-box and an

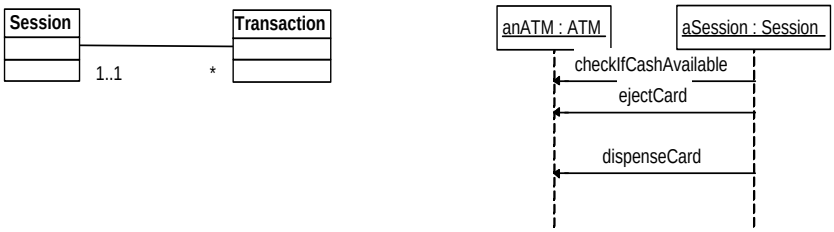



Figure 1: Classless instances conflict

instance by the concept *object*. *instance_of* specifies that an object is an instance of a certain class. *has_classmodel* is a role that contains the associated class diagram of a class. The query language of Racer can now be used to find all the classes that have no related class diagram:

$$\begin{aligned} & ans(x) \\ & \quad \leftarrow \quad class(x) \wedge object(y) \wedge instance\_of(y, x) \wedge \\ & \qquad \backslash(has\_known\_successor(x, has\_classmodel)) \end{aligned}$$

This yields the correct result, i.e. the individual $ATM$ bound to the variable $x$: $\{(ATM)\}$

### 3.2 Infinite containment

This conflict arises when the composition and inheritance relationships between classes in class diagrams form a cycle and combined with a composition relation, define a class whose instances will, directly or indirectly, be forced to contain at least one instance of the same class, causing an infinite chain of instances.

An example of this conflict is shown in Figure 2, where the class *ASCIIPrintingATM* is transitively a subclass of *ATM* and there exists a composition relation *controls* between those two classes. This composition indicates that every instance of *ATM* controls at least one instance of *ASCIIPrintingATM*. However, an instance of *ASCIIPrintingATM* is also an instance of *ATM* and as such must again control a different instance of *ASCIIPrintingATM* due to the antisymmetric property of a composition relation. (The composition relation is indicated by a black diamond.)

The direct subclass relationship is represented by the *direct_subclass* role which is a sub role of a transitive *subclass* role. A class involved in an association, is linked to this association by the role *has_association* through an association end. An association end has an aggregation kind which can be empty or an aggregation or a composite. This knowledge is represented by the role *has_aggregation* and by the concepts *aggregation* and *composition*. An association has two or more association ends, the role *ends* links the association to its ends. Each association end has a multiplicity, this is expressed by the *has_multiplicity* role. A multiplicity has a range (*has_range*) and this range has a lower and upper bound. These bounds are represented by concrete domain attributes *lower* and *upper* which have type *integer*.

The following query, expressed in nRQL, returns classes which are related by

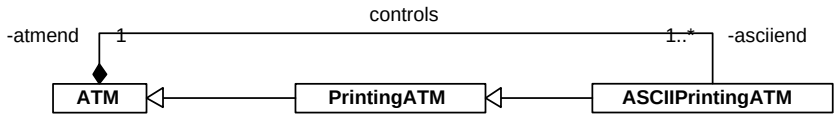



Figure 2: Infinite containment conflict

inheritance and by a composition relation introducing an infinite containment conflict:

$$
\begin{aligned}
&ans(x, y) \\
&\quad \leftarrow \quad subclass(x, y) \wedge has\_association(y, end1) \wedge \\
&\qquad\quad has\_aggregation(end1, aggreg) \wedge composition(aggreg) \wedge \\
&\qquad\quad ends(assoc, end1) \wedge ends(assoc, end2) \wedge has\_association(x, end2) \wedge \\
&\qquad\quad has\_multiplicity(end2, m2) \wedge has\_range(m2, r2) \wedge (\exists(lower).\geq_1)(r2)
\end{aligned}
$$

The result of this query asked to the A-box containing the example of Figure 2 is the answerset $\{(ASCIIPRINTINGATM, ATM)\}$.

With the current nRQL implementation, the answer sets of the queries are correct and delivered within reasonable time limits. Remark however, that this is not a mass data application, which makes this query facility suitable for our purposes.

# 4 Related Work, Discussion & Conclusion

For querying OWL semantic web repositories, the query language OWL-QL [2] has been proposed, which is the successor of the DAML+OIL query language DQL. Since OWL is basically a very expressive description logic, the proposed query language is relevant in our context as well.

An OWL-QL query is basically a full OWL KB together with a specification which of the referenced URIs in the query "body" (called query pattern in OWL terminology) are to be interpreted as variables. Variables come in three forms: *must-bind, may-bind, and do-not bind variables.* OWL-QL uses the standard notion of logical entailment: query *answers* can be seen as *logically entailed sentences* of the queried KB. Unlike in nRQL, variables cannot only be bound to constants resp. explicitly modeled A-box individuals, but also to complex *OWL terms* which are meant to denote the logically implied domain individual(s) from $\Delta^{\mathcal{I}}$. Thus, if variables in the query patterns are substituted with answer bindings, the resulting sentences are logically entailed by the queried KB. For must-bind variables, bindings *have* to be provided. May-bind variables may provide bindings or not, and *do-not-bind variables* are purely existentially quantified ("existential blanks in the query"). Moreover, OWL-QL queries are full OWL KBs, and this implies that not only extensional queries like in nRQL must be answered, but also "structural queries" are possible, such as "retrieve the subsuming concept names of the concept name father". Similar functions are also offered by Racer's API, but are not available in nRQL.

However, nRQL is not really a subset of OWL-QL. In OWL-QL, neither negation as failure nor *disjunctive A-boxes* can be expressed. Moreover, binary constraint query

atoms of nRQL as well as negated has-known-successor query atoms "are missing" in OWL-QL. The latter ones have in fact been requested by the first users of the nRQL implementation. This clearly indicates that a limited kind of *autoepistemic or closed-world query facilities* should be present in a DL query language. Negation as failure atoms are useful to measure the completeness of the current modeling in an A-box, and this is demanded by users. Thus, it might be more convincing to use a different semantics for "logical implication of queries" in the first place. Such a notion has been given in terms of the so-called $K$-operator [1], which has been used for the query language $\mathcal{ALCK}$. Roughly speaking, one could state that already *concept_instances* uses the $K$-operator in front of the concept whose instances are to be retrieved. A more detailed analysis of these relationships is left for future work.

*Horrocks and Tessaris* [5] also consider conjunctive queries for DL systems. They use two kinds of variables. Must-bind variables are called *distinguished variables*; they are bound to explicitly mentioned A-box individuals, like in nRQL. Similarly to the don't-bind variables in OWL-QL, the *non-distinguished variables* are treated as "existential blanks". Only bindings of distinguished variables are listed in the answerset of a query. Considering DLs of less expressivity than OWL, they observe that the non-distinguished variables of a query can in fact be *removed* by using a *rolling-up* technique without affecting logical entailment. For example, the query $ans(x) \leftarrow R(x,y) \wedge C(y)$ (were only $x$ is distinguished) would be rolled-up into the query $ans(x) \leftarrow \exists R.C(x)$. They also observe that for a DL which has the tree-model property and which does not offer inverse roles, a variable participating in a *join* must in fact be a distinguished variable; e.g. the variable $y$ in $ans(x) \leftarrow R(x,y) \wedge R(z,y)$.

In nRQL, the variables are always distinguished. The query $ans(x,y) \leftarrow R(x,y) \wedge C(y)$ yields $\emptyset$ over the A-box $\{\exists R.C(k)\}$. However, $ans(x) \leftarrow (\exists R.C)(x)$ could be used instead. Obviously, a rolling-up procedure could be implemented as an additional front-end processor for nRQL as well. However, another side-effect caused by the distinction of two kinds of variables is that the generation of answer tuples can no longer be understood as a simple projection. For example, the query $ans(x) \leftarrow R(x,y) \wedge C(y)$ returns $\{(k)\}$ if $x$ is distinguished and $y$ is non-distinguished, but its result is, un-intuitively, not the projection of $ans(x,y) \leftarrow R(x,y) \wedge C(y)$ to $x$, since this yields $\emptyset$ (note that $x, y$ must be both distinguished, since they appear in $ans$). Even $ans(x) \leftarrow C(x)$ returns $\emptyset$ (like *concept_instances*$(C)$), and this holds for the query language in [5] as well. For OWL-QL, however, a possible binding for $x$ might be the *concept* $C \sqcap \exists R^{-1}.\{k\}$. It should be noted that the nRQL query processing engine as well as the whole pragmatic approach for querying DL A-boxes is not dependent on any specific DL system. The same query processing engine would in fact also run with any other DL system offering A-boxes, probably without binary constraint query atoms in case the DL system does not provide concrete domains. The nRQL engine is implemented using the documented API of Racer only, and it is therefore a true "add on" whose implementation does not require any reference to the internal data structures or reasoning algorithms of Racer.

Since the beginning of the Racer endeavor, users were asking for more expressive querying facilities. The presented nRQL is a first step towards satisfying these needs. We substantiated this thesis by presenting an application from the realm of model-

based software engineering, for which it is crucial that expressive query languages are available. The new query language is an integral part since Racer-1-7-16.

## References


[1] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Adding epistemic operators to concept languages. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 342–353. Morgan Kaufmann, San Mateo, California, 1992.

[2] R. Fikes, P. Hayes, and I. Horrocks. OWL-QL - a language for deductive query answering on the semantic web. Technical Report KSL-03-14, Knowledge Systems Lab, Stanford University, CA, USA, 2003.

[3] V. Haarslev and R. Möller. Racer system description. In *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy.*, 2001.

[4] V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy*, Lecture Notes in Computer Science, pages 29–44. Springer-Verlag, June 2001.

[5] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 177–191. Springer-Verlag, 2002.

[6] Robert MacGregor and David Brill. Recognition algorithms for the LOOM classifier. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI'92)*, pages 774–779. AAAI Press/The MIT Press, 1992.

[7] Object Management Group. Unified Modeling Language specification version 1.5. formal/2003-03-01, March 2003.

[8] Ragnhild Van Der Straeten, Tom Mens, Jocelyn Simmonds, and Viviane Jonckers. Using description logic to maintain consistency between UML models. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, pages 326–340. Springer, 2003.

[9] Ragnhild Van Der Straeten, Jocelyn Simmonds, and Tom Mens. Detecting inconsistencies between UML models using description logic. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *Proceedings of the 2003 International Workshop on Description Logics (DL2003), Rome, Italy September 5-7, 2003*, volume 81 of *CEUR Workshop Proceedings*, pages 260–264, 2003.

# A Uniform Tableaux-Based Approach to Concept Abduction and Contraction in $\mathcal{ALN}$

S. Colucci[1], T. Di Noia[1], E. Di Sciascio[1],
F.M. Donini[2], M. Mongiello[1]
1: Politecnico di Bari, BARI, Italy
{s.colucci,t.dinoia,disciascio,mongiello}@poliba.it
2: Università della Tuscia, VITERBO, Italy donini@unitus.it

**Abstract**

We present algorithms based on truth-prefixed tableaux to solve both Concept Abduction and Contraction in $\mathcal{ALN}$ DL. We also analyze the computational complexity of the problems, showing that the upper bound of our approach meets the complexity lower bound. The work is motivated by the need to offer a uniform approach to reasoning services useful in semantic-based matchmaking scenarios.

## 1 Motivation

In recent papers [16, 15], Description Logics (DLs) have been proposed to model knowledge domains in Semantic Web scenarios. A challenging issue in such scenarios is the matchmaking problem which is finding an offered resource described by a formalism with an unambiguous semantics [21, 8, 17]. Using DLs to describe resources, it is possible to infer which of them satisfies the request either completely (*i.e.*, subsumes the request) or potentially (*i.e.*, the conjunction of the requested resource and the offered one is satisfiable) or partially(*i.e.*, the conjunction of the requested resource and the offered one is not satisfiable).

In [9, 7] Concept Abduction and Concept Contraction have been proposed as non-standard inference services in DL, to capture in a logical way the reasons why a resource $S_1$ should be preferred to another resource $S_2$ for a given request $D$, and vice versa. Although efficient reasoning methods based on tableaux have been successfully implemented for standard inference services in DL —satisfiability, subsumption, instance check, etc. [1, Ch.8-9] — non-standard reasoning services have been usually solved by different methods, such as automata[1, Ch.6], making a complete system built on heterogeneous technologies. Such an approach leads to incomparable optimization techniques and partial duplication of services — *e.g.*, a module computing *least common subsumer* computes also subsumption. This motivates our research in tableaux-based methods for Concept Abduction and Concept Contraction.

Related work on abduction using tableaux is in [6], where tableaux are used for the multimodal logic **K**, corresponding to the DL $\mathcal{ALC}$. However, in that work the purpose was not to find efficient methods, and contraction was not considered. Here we devise more efficient methods, for a $\mathcal{ALN}$ logic, which would correspond to a syntactically-restricted modal logic with graded modalities.

## 2 Abduction and Contraction in $\mathcal{ALN}$

We start with a few definitions of the problem and then move on to discuss its computational complexity. We assume the reader is familiar with DLs, and refer to [1] for a thourough introduction to $\mathcal{ALN}$, TBoxes, satisfiability, subsumption (denoted as $\sqsubseteq$) and subsumption w.r.t. a TBox $\mathcal{T}$ (denoed as $\sqsubseteq_{\mathcal{T}}$).

Here we deal only with a simple form of axioms in the TBox, where in the left hand side of inclusions only concept names can appear and with each concept name as at most one left-hand side of an axiom. Moreover, we admit only *acyclic* TBoxes, in the following sense.

**Definition 1 (Dependency Graph of a TBox)** Let $\mathcal{T}$ be a TBox. The dependency graph of $\mathcal{T}$ is a graph $\mathcal{G}_{\mathcal{T}} = (N, V)$ whose nodes are concept names, and whose arcs are defined as follows: if $A \sqsubseteq C \in \mathcal{T}$, and concept name $B$ appears in $C$, then there is an arc from node $A$ to node $B$.

A TBox $\mathcal{T}$ is said to be *acyclic* if $\mathcal{G}_{\mathcal{T}}$ contains no cycles. Even for this simple form of acyclic TBox, it is known that subsumption is coNP-hard [18] and also satisfiability is coNP-hard [5, 4]. However, all hardness reductions rely on "deep" TBoxes — TBoxes in which the length of the longest path in $\mathcal{G}_{\mathcal{T}}$ is allowed to grow as large as $O(|\mathcal{T}|)$. For$\mathcal{ALN}$ TBoxes that, in Nebel's words [18], are "bushy but not deep", satisfiability and subsumption can be solved in polynomial time [3].

**Definition 2 (Bushy TBox)** A sequence of acyclic TBoxes $\mathcal{T}_1, \ldots, \mathcal{T}_n, \ldots$ are *bushy* if the size of the longest path in $\mathcal{G}_{\mathcal{T}_i}$ is bounded by $O(\log |\mathcal{T}_i|)$.

In the rest of the paper, we limit our attention to bushy TBoxes in $\mathcal{ALN}$.

### 2.1 Concept Abduction in $\mathcal{ALN}$

We follow the notation in [9, 7], excluding the choice of the DL which in our case is always $\mathcal{ALN}$.

**Definition 3** Let $C$, $D$, be two concepts in $\mathcal{ALN}$, and $\mathcal{T}$ be a set of axioms in $\mathcal{ALN}$, where both $C$ and $D$ are satisfiable in $\mathcal{T}$. A *Concept Abduction Problem* (CAP), denoted as $\langle C, D, \mathcal{T}\rangle$, is finding a concept $H \in \mathcal{ALN}$ such that $\mathcal{T} \not\models C \sqcap H \equiv \bot$, and $\mathcal{T} \models C \sqcap H \sqsubseteq D$.

We use $\mathcal{P}$ as a symbol for a CAP, and we denote with $SOLCAP(\mathcal{P})$ the set of all solutions to a CAP $\mathcal{P}$. For $SOLCAP(\mathcal{P})$ the three following minimality criteria have been proposed.

**Definition 4** Let $\mathcal{P} = \langle C, D, \mathcal{T}\rangle$ be a CAP. The set $SOLCAP_{\sqsubseteq}(\mathcal{P})$ is the subset of $SOLCAP(\mathcal{P})$ whose concepts are maximal under $\sqsubseteq_{\mathcal{T}}$. The set $SOLCAP_{\leq}(\mathcal{P})$ is the subset of $SOLCAP(\mathcal{P})$ whose concepts have minimum length. The set $SOLCAP_{\sqcap}(\mathcal{P})$ is the subset of $SOLCAP(\mathcal{P})$ whose concepts are minimal conjunctions, *i.e.*, if $C \in SOLCAP_{\sqcap}(\mathcal{P})$ then no sub-conjunction of $C$ is in $SOLCAP(\mathcal{P})$. We call such solutions *irreducible abductions*.

The three forms of minimality are related by: both $SOLCAP_{\sqsubseteq}(\mathcal{P})$ and $SOLCAP_{\leq}(\mathcal{P})$ are included in $SOLCAP_{\sqcap}(\mathcal{P})$ [9, Prop.2].

## 2.2 Concept Contraction in $\mathcal{ALN}$

As defined by Gärdenfors' [12], who formalized the revision of a knowledge base $\mathcal{K}$ with a new piece of knowledge $A$, is made up of *(i)* a *contraction* operation, which results in a new knowledge base $\mathcal{K}^-_A$ such that $\mathcal{K}^-_A \not\models \neg A$, *(ii)* the conjunction of $A$ to $\mathcal{K}^-_A$.

**Definition 5** Let $C$, $D$, be two concepts in $\mathcal{ALN}$, and $\mathcal{T}$ be a set of axioms in $\mathcal{ALN}$, where both $C$ and $D$ are satisfiable in $\mathcal{T}$. A *Concept Contraction Problem* (CCP), denoted as $\langle C, D, \mathcal{T}\rangle$, is finding a pair of concepts $\langle G, K\rangle$ (both in $\mathcal{ALN}$) such that $\mathcal{T} \models C \equiv G \sqcap K$, and $K \sqcap D$ is satisfiable in $\mathcal{T}$. We call $K$ a *contraction* of $C$ according to $D$ and $\mathcal{T}$.

Also for Concept Contraction, one is interested in a minimal contraction, according to some form of minimality.

**Definition 6** Let $\mathcal{Q}$ =$\langle C, D, \mathcal{T}\rangle$ be a CCP. The set $SOLCCP_{\sqsubseteq}(\mathcal{Q})$ is the subset of solutions $\langle G, K\rangle$ in $SOLCCP(\mathcal{Q})$ such that $G$ is maximal under $\sqsubseteq_{\mathcal{T}}$. The set $SOLCCP_{\leq}(\mathcal{Q})$ is the subset of $SOLCCP(\mathcal{Q})$ such that $G$ has minimum length. The set $SOLCCP_{\sqcap}(\mathcal{Q})$ is the subset of $SOLCCP(\mathcal{Q})$ whose concepts are minimal conjunctions, *i.e.*, if $\langle G, K\rangle \in SOLCCP_{\sqcap}(\mathcal{Q})$ then no sub-conjunction $G'$ of $G$ is such that $\langle G', K'\rangle \in SOLCCP(\mathcal{Q})$ for any $K'$. We call such solutions *irreducible contractions*.

We now analyze the complexity of computing a minimum-length concept abduction in $\mathcal{ALN}$. Proposition 3 in [9] yields a trivial polynomial-time lower bound for Concept Abduction in $\mathcal{ALN}$ with a bushy TBox. Using a simple reduction, we show a tighter lower bound, using an elementary form of Tbox: the problem is NP-hard. It is sufficient to have a constant-depth concept hierarchy — *i.e.*, a set of inclusions between concept names where the longest path in $\mathcal{G}_{\mathcal{T}}$ has length 1 — to model the set-covering model for abduction [19].

**Definition 7 (Set Covering)** Let $U = \{a_1, \ldots, a_n,\}$ be a set, let $s_1, \ldots, s_m$, be a collection of subsets of $U$ such that $\cup_i s_i = U$ and let $k \leq n$ be an integer. The *Set covering* problem is deciding whether there exists a subcollection of subsets $s_{i_1}, \ldots, s_{i_k}$ whose union covers $U$.

**Theorem 1** Minimal-length Concept Abduction in $\mathcal{ALN}$ is NP-hard, even when $\mathcal{T}$ is a bushy concept hierarchy.

Given an instance of Set Covering, we construct a CAP $\mathcal{P}$ =$\langle C, D, \mathcal{T}\rangle$ as follows. Let $A_1, \ldots, A_n, B_1, \ldots, B_n$ be $2n$ concept names, where each $A_i$ and $B_i$ is one-one with $a_i$, and let $S_1, \ldots, S_m$, be also concept names, one-one with subsets of $U$. Let the Tbox $\mathcal{T}$ be defined as follows: $\{S_i \sqsubseteq A_j, S_i \sqsubseteq B_j | a_j \in s_i\}$. Now we prove that $s_{i_1}, \ldots, s_{i_k}$ is a minimal set covering iff $S_{i_1} \sqcap \cdots \sqcap S_{i_k} \in SOLCAP_{\leq}(\mathcal{P})$, where $C = \top$ and $D = A_1 \sqcap \cdots \sqcap A_n \sqcap B_1 \sqcap \ldots \sqcap B_n$. First of all, we prove a property of this construction.

**Property 1** Every minimal-length abduction $H$ of $\mathcal{P}$ contains neither $A_i$ nor $B_i$, for every $i = 1, \ldots, n$.

*Proof.* Let $H \in SOLCAP(\mathcal{P})$ and suppose $A_3$ – say – is a conjunct of $H$. If there is a concept $S$ in $H$, such that $S \sqsubseteq A_3 \in \mathcal{T}$, then $H$ without $A_3$ is a shorter abduction. Otherwise, since$C \sqcap H \equiv H \sqsubseteq D$, also $B_3$ must be a conjunct of $H$. In this case, let $S$ be a

concept such that $S \sqsubseteq A_3, S \sqsubseteq B_3 \in \mathcal{T}$. Then the concept$H$ without $A_3, B_3$ and with $S$ is a solution one conjunct shorter. The same line of reasoning could be repeated if a concept $B$ is a conjunct of $H$. Therefore, every minimal-length abduction contains neither $A_i$ nor $B_i$, for every $i = 1, \ldots, n$. □

(If) Suppose $s_{i_1}, \ldots, s_{i_k}$ is a set covering. Then, $H \doteq S_{i_1} \sqcap \cdots \sqcap S_{i_k}$ is such that $C \sqcap H$ is satisfiable (in fact, every conjunction is satisfiable in this CAP), and $C \sqcap H \sqsubseteq D$. Moreover, if $H$ is not a minimal-length abduction, then let $H' \in SOLCAP_{\leq}(\mathcal{P})$. For the above property, $H'$ does not contain $A$'s and $B$'s. Then it is straightforward to define a shorter set covering from $H'$,contradicting the fact that $s_{i_1}, \ldots, s_{i_k}$ was a minimal set covering. (Only-if) On the other hand, suppose $H \in SOLCAP_{\leq}(\mathcal{P})$. Then $H$ does not contain $A$'s and $B$'s, so it can be written as $S_{i_1} \sqcap \cdots \sqcap S_{i_k}$, which identifies a collection of subsets $\mathcal{S}_H = s_{i_1}, \ldots, s_{i_k}$. Since $H \sqsubseteq D$, also $\mathcal{S}_H$ covers $U$; moreover, if $S_H$ was not minimal, it would define a shorter solution for $\mathcal{P}$, contradicting the hypothesis. □

We observe that a (more realistic) CAP allows one to put weights and probabilities attached to concepts in order to measure the importance that a user gives to a specified characteristic. Obviously, also this weighted version of CAP is NP-hard.

## 3 Calculus and Algorithms

In the following we assume the reader be familiar with tableaux (***e.g.***, [14]). In this section two algorithms working on tableaux for $\mathcal{ALN}$ concepts are presented. They both use the same set of rules: the first one (*contract*) computes a solution $\langle G, K \rangle$ for a CCP, the second one (*abduce*) solves a CAP computing $H$.
Tableaux for DLs use a labeling function $\mathcal{L}$ to map an individual $x$ to a set of concepts $\mathcal{L}(x)$ such that for every concept $C$, $C \in \mathcal{L}(x)$ stands for the formula $C(x)$, and similarly for roles $R \in \mathcal{L}(x, y)$. Here we distinguish between formulas labeled "true" and formulas labeled "false" in the tableaux[20], hence we use two labeling functions **T**() and **F**(), both going from individuals to sets of concepts, and from pairs of individuals to sets of roles. A (usual) tableau branch is now represented by two functions **T**() and**F**(). Moreover, we write in the name of an individual $x$ its history, ***i.e.***, the string identifying $x$ is made up of integers and role symbols, such as $x = 1R3Q7$, which means that individual $x$ is used for concepts in a quantification involving role $R$, and inside, a quantification involving role $Q$. Integers in between roles make sure that such strings are unique, ***i.e.***, there can be two individuals with the same role sequence, but not with the same integer sequence[11].

Given an individual $x$ in a tableau, an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies two tableau labels $\mathbf{T}(x)$ and $\mathbf{F}(x)$ if, for every concept $C \in \mathbf{T}(x)$ and every concept $D \in \mathbf{F}(x)$, it is $x^{\mathcal{I}} \in C^{\mathcal{I}}$ and $x^{\mathcal{I}} \notin D^{\mathcal{I}}$ respectively. Similarly, $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies two tableau labels $\mathbf{T}(x, y)$ and $\mathbf{F}(x, y)$ if for every role $R \in \mathbf{T}(x, y)$ and for every role $Q \in \mathbf{F}(x, y)$ it holds $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $(x^{\mathcal{I}}, y^{\mathcal{I}}) \notin Q^{\mathcal{I}}$. We note that for $\mathcal{ALN}$ DL, every role $Q$ appearing in a label $\mathbf{F}(x, y)$ is of the form $\neg R$, hence $Q \in \mathbf{F}(x, y)$ means, in fact, $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$ too. An interpretation satisfies a tableau branch if it satisfies $\mathbf{T}(x)$, $\mathbf{F}(x)$, $\mathbf{T}(x, y)$ and $\mathbf{F}(x, y)$ for every individual $x$, and for every pair of individuals $x, y$ in the branch.

We assume that concepts are always simplified in Negation Normal Form (NNF, see [1,

ch.2]), so that negations come only in front of concept names. Observe that for $C \in \mathcal{ALN}$, $\overline{C}$ may not belong to $\mathcal{ALN}$ since it is not closed under negation. In what follows, given a concept $C$, we denote with $\overline{C}$ the NNF of $\neg C$. Rules come in pairs, first the (usual) version with a construct in the **T**-constraints, then the dual construct in the **F**-constraints. However, groups 2 and 3 have only **F**-constraints because the correspondent formulae do not appear in our tableaux for $\mathcal{ALN}$.

1. conjunctions:

   **T⊓)** if $C \sqcap D \in \mathbf{T}(x)$, then add both $C$ and $D$ to $\mathbf{T}(x)$.

   **F⊔)** if $C \sqcup D \in \mathbf{F}(x)$, then add both $C$ and $D$ to $\mathbf{F}(x)$.

2. disjunctions (branching rules):

   **F⊓)** if $C \sqcap D \in \mathbf{F}(x)$, then add either $C$ or $D$ to $\mathbf{F}(x)$.

3. existential quantifications:

   **F∀)** if $\forall R.C \in \mathbf{F}(x)$, then pick up a new individual $y = x \circ R \circ m$ (where $m$ is an integer such that $y$ is unique), add $\neg R$ to $\mathbf{F}(x, y)$, and let $\mathbf{F}(y) := \{C\}$.

4. universal quantifications:

   **T∀)** if $\forall R.C \in \mathbf{T}(x)$ and there exists an individual $y$ such that either $R \in \mathbf{T}(x, y)$, or $\neg R \in \mathbf{F}(x, y)$, then add $C$ to $\mathbf{T}(y)$.

   **F∃)** if $\exists R.C \in \mathbf{F}(x)$, and there exists an individual $y$ such that either $R \in \mathbf{T}(x, y)$, or $\neg R \in \mathbf{F}(x, y)$, then add $C$ to $\mathbf{F}(y)$.

5. at-least number restrictions:

   **T⩾)** if $\geqslant n\, R \in \mathbf{T}(x)$, with $n > 0$, and for every individual $y$ neither $R \in \mathbf{T}(x, y)$ nor $\neg R \in \mathbf{F}(x, y)$, then pick up a new individual $y = x \circ R \circ m$ (where $m$ is an integer such that $y$ is unique), add $R$ to $\mathbf{T}(x, y)$, and let $\mathbf{T}(y) := \emptyset$.

   **F⩽)** if $\leqslant n\, R \in \mathbf{F}(x)$ and for every individual $y$ neither $R \in \mathbf{T}(x, y)$ nor $\neg R \in \mathbf{F}(x, y)$, then pick up a new individual $y = x \circ R \circ m$ (where $m$ is an integer such that $y$ is unique), add $\neg R$ to $\mathbf{F}(x, y)$, and let $\mathbf{F}(y) := \emptyset$.

6. at-most number restrictions:

   **T⩽)** if $\leqslant 1\, R \in \mathbf{T}(x)$, and there are 2 individuals $y_1, y_2$ such that for $i \in 1, 2$ it is either $R \in \mathbf{T}(x, y_i)$ or $\neg R \in \mathbf{F}(x, y_i)$, then let $\mathbf{T}(y_1) := \mathbf{T}(y_1) \cup \mathbf{T}(y_2)$, let $\mathbf{F}(y_1) := \mathbf{F}(y_1) \cup \mathbf{F}(y_2)$, and eliminate $y_2$ in the branch.

   **F⩾)** if $\geqslant 2\, R \in \mathbf{F}(x)$ and there are 2 individuals $y_1, y_2$ such that for $i \in 1, 2$ it is either $R \in \mathbf{T}(x, y_i)$ or $\neg R \in \mathbf{F}(x, y_i)$, then let $\mathbf{T}(y_1) := \mathbf{T}(y_1) \cup \mathbf{T}(y_2)$, let $\mathbf{F}(y_1) := \mathbf{F}(y_1) \cup \mathbf{F}(y_2)$, and eliminate $y_2$ in the branch.

7. axioms in $\mathcal{T}$:

**F$\sqsubseteq$)** if $x$ is an individual such that either $A \in \mathbf{T}(x)$ or $\neg A \in \mathbf{F}(x)$ in the branch, and $A \sqsubseteq C \in \mathcal{T}$, then add $A \sqcap \overline{C}$ to $\mathbf{F}(x)$.

**F$\doteq_1$)** if $x$ is an individual such that either $A \in \mathbf{T}(x)$ or $\neg A \in \mathbf{F}(x)$ in the branch, and $A \doteq C \in \mathcal{T}$, then add $A \sqcap \overline{C}$.

**F$\doteq_2$)** if $x$ is an individual such that either $\neg A \in \mathbf{T}(x)$ or $A \in \mathbf{F}(x)$ in the branch, and $A \doteq C \in \mathcal{T}$, then add $C \sqcap \neg A$ to $\mathbf{F}(x)$.

When more than one rule can be applied, we always give *lowest* precedence to Rules **T**$\geqslant$) and **F**$\leqslant$), while other rules can be applied in any order. In group 7 (axioms in $\mathcal{T}$) a *lazy unfolding* of the TBox is taken into account [2, 13]. Following this strategy, axioms in $\mathcal{T}$ are dealt in a deterministic manner avoiding the exponential increase in the search space due to the non-deterministic choices in a pure-tableau approach.

We now split the definition of clash (an explicit inconsistency) between clashes involving the same truth prefix (homogeneous clashes) and those involving both prefixes (heterogeneous clashes).

**Definition 8 (Clash)** *A branch contains a* homogeneous clash *if it contains one of the following:*

1. *either $\bot \in \boldsymbol{T}(x)$ or $\top \in \boldsymbol{F}(x)$, for some individual $x$;*
2. *either $A, \neg A \in \boldsymbol{T}(x)$ or $A, \neg A \in \boldsymbol{F}(x)$ for some individual $x$ and some concept name $A$;*
3. *either $\geqslant n\, R, \leqslant m\, R \in \boldsymbol{T}(x)$ with $m < n$, or $\leqslant n\, R, \geqslant m\, R \in \boldsymbol{F}(x)$ with $m-1 < n+1$, for some individual $x$, and some role name $R$.*

*A branch contains a* heterogeneous clash *if it contains one of the following:*

1. *$\boldsymbol{T}(x) \cap \boldsymbol{F}(x)$ contains either $A$ or $\neg A$ for some individual $x$ and some concept name $A$;*
2. *either $\geqslant n\, R \in \boldsymbol{T}(x)$ and $\geqslant m\, R \in \boldsymbol{F}(x)$ with $m - 1 < n$, or $\leqslant n\, R \in \boldsymbol{T}(x)$ and $\leqslant m\, R \in \boldsymbol{F}(x)$ with $n < m + 1$, for some individual $x$, and some role $R$*

A branch is *complete* if no new rule application is possible to labels in the branch. A complete branch is *open* if it contains no clash, otherwise it is *closed*. A complete tableau is open if it contains at least one open branch, otherwise it is closed. We call a branch with a homogeneous clash *as good as complete*. Soundness and completeness of the calculus follow from the version without prefixes [10].

**Theorem 2** Let $C, D$ be two concepts in $\mathcal{ALN}$, and $\mathcal{T}$ an acyclic TBox in $\mathcal{ALN}$. Then $C \sqsubseteq D$ in $\mathcal{T}$ iff the tableau starting from $C \in \mathbf{T}(x), D \in \mathbf{F}(x)$ is closed.

Moreover, with prefixed tableaux we can distinguish between "real"subsumption, and subsumption stemming from inner contradiction in concepts.

**Theorem 3** Let $C, D$ be two concepts in $\mathcal{ALN}$, and $\mathcal{T}$ an acyclic TBox in $\mathcal{ALN}$. If every branch of the tableau starting from $C \in \mathbf{T}(1), D \in \mathbf{F}(1)$ contains a homogeneous clash, then either $C \equiv \bot$ or $D \equiv \top$ in $\mathcal{T}$.

We now present the two algorithms for Concept Contraction and Concept Abduction, that need some preliminary definitions.

Both algorithms use a function $roles(x)$, that given an individual $x$ (as a sequence of integers and roles) returns the sequence of roles in $x$ (without integers). For example, $roles(1R3Q7) = RQ$. We let $roles(k) = \varepsilon$, *i.e.*, when $x$ is just one integer, $roles(x)$ returns the empty sequence. For a given concept $C$, and a sequence of roles $\sigma$, we define $\forall\sigma.C$ as $\forall R_1.(\cdots(\forall R_n.C)\cdots)$ if $\sigma = R_1 \cdots R_n$, and $\forall\sigma.C \doteq C$ in the special case in which $\sigma = \varepsilon$.

Moreover, we assume that atomic concepts (names and number restrictions) can be given a unique index, as in $A^1 \sqcap \forall R.((\leqslant 1\, Q)^2 \sqcap A^3)$. Hence the substitution of an *occurence* of a concept can be defined: we let $D[C \rightarrow \top]$ denote the substitution of an occurrence of an indexed atomic concept $C$ with the concept $\top$, inside a concept $D$. For example, if $D$ is the concept above, then $D[A^1 \rightarrow \top] = \top \sqcap \forall R.((\leqslant 1\, Q)^2 \sqcap A^3)$, while $D[A^3 \rightarrow \top] = A^1 \sqcap \forall R.((\leqslant 1\, Q)^2 \sqcap \top)$. Multiple substitutions are denoted by a set of concepts, *e.g.*, if $\mathcal{G} = \{A, B\}$ then $D[C \rightarrow \top]_{C\in\mathcal{G}}$ means $(D[A \rightarrow \top])[B \rightarrow \top]$. Observe that since we substitute only atomic concepts, the order of substitutions is ininfluent. For both algorithms, we assume that concepts are indexed, so that substitutions are unambiguous.

**Algorithm** *contract*
**input:** $\mathcal{ALN}$ concepts $C$, $D$, acyclic TBox $\mathcal{T}$
**output:** concepts $K$ (keep), $G$ (giveup)
**begin**
  compute a complete tableau $\tau$ for $\mathcal{T}$, $D \in \mathbf{T}(x), \overline{C} \in \mathbf{F}(x)$
  **if** $\tau$ is open **then**
    /* no contraction needed */
    **return** $G := \top, K := D$
  **else if** every branch in $\tau$ contains a homogeneous clash **then**
    /* either $C$ or $D$ is unsatisfiable in $\mathcal{T}$ */
    **return** fail
  **else**
    choose(*) a branch $\beta$ containing only heterogeneous clashes;
    **let** $\mathcal{G} := \{\langle C_i, x_i\rangle | C_i \in \mathbf{T}(x_i), \overline{C_i} \in \mathbf{F}(x_i)$ is a clash in $\beta\}$
    **let** $G := \sqcap_{\langle C_i,x_i\rangle\in\mathcal{G}} \forall roles(x_i).C_i$
    **let** $K := D[C_i \rightarrow \top]_{\langle C_i,x_i\rangle\in\mathcal{G}}$
    **return** $G, K$
**end**

Observe that the algorithm *contract* contains a choice in step (*). This choice is needed to select the contraction according to some minimality criterion. Only branches without homogeneous clashes need to be completely expanded, even after the first clash has been found. Observe also that substituting an occurrence of a concept $C$ with $\top$ corresponds, in $\mathcal{ALN}$, to eliminating the occurrence. We preferred this notation instead of eliminating occurrences, since it appears more concise.

**Theorem 4** The concepts $G, K$ returned by the Algorithm *contract* are a Contraction of $D$ w.r.t. $C$ and $\mathcal{T}$.

*Proof.* First, note that $K \sqcap C$ is satisfiable by definition of $K$; in fact, the tableau for $K \sqcap C$ is the same as the tableau for $D \sqcap C$, but it has now at least one open branch $\beta$, in which all clashes have been removed. Secondly, $D \equiv G \sqcap K$ by construction. □

Note that Algorithm $contract$ proves that Concept Contraction in $\mathcal{ALN}$ with bushy TBoxes is solvable in polynomial time.

We now present the algorithm for Concept Abduction, which also uses the tableaux rules previously defined.

**Algorithm** $abduce$
**input:** $\mathcal{ALN}$ concepts $C$, $D$, acyclic TBox $\mathcal{T}$
**output:** concept H (hypotheses)
**begin**
compute a complete tableau $\tau$ for $\mathcal{T}$, $C \in \mathbf{T}(x), D \in \mathbf{F}(x)$
**if** $\tau$ is closed **then**
/* no abduction needed */
**return** $H := \top$
**else**
choose(*) a set of pairs $\mathcal{H} := \{\langle C_i, x_i \rangle\}$ and
**let** $H := \sqcap_{\langle C_i, x_i \rangle \in \mathcal{H}} \forall roles(x_i).C_i$
such that (1) every open branch in $\tau$ contains at least
one constraint $C_i \in \mathbf{F}(x_i)$ from $\mathcal{H}$
(2) $C \sqcap H$ is satisfiable in $\mathcal{T}$
**return** $H$
**end**

**Theorem 5** The concept $H$ returned by the Algorithm $abduce$ is a solution of the CAP $\langle C, D, \mathcal{T} \rangle$.

*Proof.* Let $\tau$ be the tableau built by *abduce*. The tableau starting from $C \sqcap H \in \mathbf{T}(1), D \in \mathbf{F}(1)$ is $\tau$, plus the constraints signed **T** from $H$. Hence, it is closed. Hence, $\mathcal{T} \models C \sqcap H \sqsubseteq D$. Regarding the condition $C \sqcap H$ satisfiable in $\mathcal{T}$, it is enforced by Condition (2) in the choice of $\mathcal{H}$. □

Condition (2) is necessary in $abduce$, since heterogeneous clashes could be formed also by contradicting an axiom in $\mathcal{T}$. In that case, although it still holds $C \sqcap H \sqsubseteq D$ in $\mathcal{T}$, the subsumption trivially holds since $C \sqcap H \equiv \bot$. We conclude the section by showing that our Algorithm $abduce$ puts an upper bound to Concept Abduction that meets the lower bound proved in the previous section.

**Theorem 6** Let $\mathcal{P} = \langle C, D, \mathcal{T} \rangle$ a Concept Abduction Problem, where $C, D$ are concepts in $\mathcal{ALN}$, $\mathcal{T}$ is a bushy TBox in $\mathcal{ALN}$ and k is an integer. Deciding whether there exists a solution of length $k$ in $SOLCAP_{\leq}(\mathcal{P})$ is NP-complete.

*Proof.* Hardness was shown in Thm. 1. Membership in NP is proved by the correctness of Algorithm $abduce$, since it is sufficient to run the algorithm, and guessing in the nondeterministic step (*) a set $\mathcal{H}$ that defines a concept $H$ of length $k$. □

## 4 Conclusion

We have shown how Concept Abduction and Concept Contraction for DL $\mathcal{ALN}$ can be performed using prefixed-tableaux. For such DL, we proved optimality of the methods by showing that they meet lower bounds obtained by a complexity analysis. Although devised for a simple DL, we believe that the proposed approach could be easily extended to more expressive DLs.

### Acknowledgements

We wish to thank A. Calí and D. Calvanese for fruitful discussions and the anonymous reviewers for helping in improving paper quality. This work was carried out in the framework of projects PON CNOSSO and MS3DI.

## References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel Schneider (eds.). *The Description Logic Handbook*. Cambridge University Press, 2003.

[2] F. Baader, B. Hollunder, B. Nebel, H. Profitlich, and E. Franconi. An empirical analysis of optimization techniques for terminological representation systems or "making KRIS get a move on". In *KR'92*, pages 270–281.

[3] A. Borgida and P. F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[4] Martin Buchheit, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. A refined architecture for terminological systems: Terminology = schema + views. *Artif. Intell.*, 99(2):209–260, 1998.

[5] D. Calvanese. Reasoning with inclusion axioms in description logics. In *ECAI'96*, pages 303–307. John Wiley & Sons, 1996.

[6] M. Cialdea Mayer and F. Pirri. Modal propositional abduction. *Journal of the IGPL*, 3(6):99–117, 1995.

[7] S. Colucci, T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Concept abduction and contraction in description logics. In *DL 2003*, 2003.

[8] T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. A system for principled matchmaking in an electronic marketplace. In *WWW'03*. ACM, 2003.

[9] T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Abductive matchmaking in description logics. In *IJCAI 2003*, 2003.

[10] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134:1–58, 1997.

[11] F.M. Donini and F. Massacci. Exptime tableaux for $\mathcal{ALC}$. *Artif. Intell.*, 124:87–138, 2000.

[12] P. Gardenfors. *Knowledge in Flux*. Mit Press, Bradford Book, 1988.

[13] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *KR'98*, pages 636–645.

[14] I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In *IJCAI 2001*, pages 199–204, 2001.

[15] I. Horroks, P. Patel-Schneider, and F. van Harmelen. From shiq to rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[16] M. Klein, J. Broekstra, D. Fensel, F. van Harmelen, and I. Horrocks. *Ontologies and schema languages on the web*. MIT Press, 2003.

[17] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW'03*. ACM Press, 2003.

[18] B. Nebel. Terminological reasoning is inherently intractable. *Artif. Intell.*, 43:235–249, 1990.

[19] J.A. Reggia, D.S. Nau, and Y. Wang. Diagnostic expert systems based on a set covering model. *International Journal on Man Machine Studies*, 19:437–460, 1983.

[20] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, New York inc., 1968.

[21] D. Trastour, C. Bartolini, and C. Priest. Semantic web support for the business-to-business e-commerce lifecycle. In *WWW'02*, pages 89–98. ACM, 2002.

# Tableau Systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$

Jan Hladik*, Jörg Model
Chair for Automata Theory, TU Dresden
{hladik,model}@tcs.inf.tu-dresden.de

## 1 Introduction

Two prominent families of algorithms for the satisfiability test of DLs are *automata-based algorithms* (see e.g. [6]), which translate a concept $C$ into an automaton $A_C$ accepting all (abstractions of) models for $C$, and *tableau algorithms* (TAs) [2], which incrementally create a tree-shaped (pre-) model for $C$ using a set of *completion rules*. In short, the advantages of automata algorithms are on the theoretical side, because in many cases the proofs are very elegant and provide tight complexity bounds (in particular for ExpTime-complete logics), whereas the advantages of tableau algorithms are on the practical side, since they are well suited for implementation and optimisation.

Thus, an approach combining both advantages is highly desirable. In [1], we introduced *tableau systems (TSs)*, a framework for tableau algorithms. From a TS for a DL $\mathcal{L}$, we can derive an automata algorithm deciding satisfiability of $\mathcal{L}$ inputs in exponential time, and a tableau algorithm for $\mathcal{L}$, including an appropriate blocking condition which ensures termination. As an application of this framework, we present in this paper tableau systems for two expressive DLs, the new DL $\mathcal{SHIO}$ and the well-known $\mathcal{SHIQ}$ [5]. Our main results are the following: firstly, we obtain that $\mathcal{SHIO}$ satisfiability is ExpTime-complete and can be decided by a tableau algorithm. Secondly, we will see that although these two logics share most of their constructs, they lead to quite different TSs, which demonstrates how the capabilities of our framework can be used to capture different language properties. Thirdly, the succinctness of the proofs demonstrates how our framework simplifies the design of TAs.

## 2 The Tableau Framework for ExpTime Logics

Although the term "DL tableau algorithm" is not formally defined, the following features can be considered as the common ground for existing algorithms: a TA operates on a *completion tree* which represents a model for the input (e.g. a concept term, possibly together with an RBox or TBox) under consideration. To generate this model, the TA starts with an initial tree, which is subsequently modified according to a set of *completion rules*, which may or may not be applicable to a certain node. These rules essentially describe subtrees of the completion tree before and after rule application, i.e. pre- and post-conditions. In many cases, they only operate on a node and its direct neighbours, but sometimes they also consider nodes which are arbitrarily

*The first author of this paper is supported by the DFG, Project No. GR 1324/3-4.

far apart (e.g. for nominals as in [4]) or global information which is relevant for all nodes (e.g. for concept or role inclusion axioms). In general, it is possible that several rules are applicable to a node at the same time, but the sequence of rule applications is *don't-care*-nondeterministic, i.e. every sequence will lead to the same result. In contrast, some rules, e.g. for disjunction, are *don't-know*-nondeterministic, i.e. it is possible that one disjunct may lead to a model, while another one may not.

An inconsistency in the generated completion tree is detected through *clash triggers*, subtrees containing an obvious contradiction, which means that a completion tree containing a clash trigger cannot be transformed into a model. Thus, a model is represented by a completion tree which is *saturated*[1], i.e. to which no rule is applicable, and *clash-free*, i.e. not containing a clash trigger.

In our framework, we formalise a completion tree as an *S-tree* $((V, E, n, \ell), \mu)$, where $(V, E)$ is a bounded width tree, $n$ and $\ell$ are node and edge labelling functions, and $\mu$ is a *global memory* which is used to store information relevant for all nodes. Rules are are represented by *S-Patterns*, which are essentially S-trees of bounded depth. For every S-pattern $P$, we describe the possible modifications by *all* rules as follows: $P$ is mapped to a set of sets of patterns $\{S_1, S_2, \ldots, S_n\}$, where the choice of the set $S_i$ represents the don't-care choice of the rule that is applied, and the choice of the pattern within the set $S_i$ represents the don't-know choice of the alternative (a deterministic rule corresponds to a singleton set $S_i$). In particular, if $P$ is saturated, then it is mapped to the empty set. As an example, consider a pattern $P$ in which the node $n$ is labelled with $\{C \sqcap D, E \sqcup F\}$. Here, one can define the rules as follows: $P \mapsto \{\{P_1\}, \{P_2, P_3\}\}$, and in $P_1$, $C$ and $D$ are added to the label of $n$, whereas $E$ is added in $P_2$ and $F$ in $P_3$.

S-patterns are also used to describe clash triggers: within our framework, a clash trigger is simply a pattern which contains a contradiction (in the context of the logic under consideration). Note that S-patterns also contain a $\mu$ component, thus rules and clash triggers can read the global memory, and rules can also modify it.

Since our intention was to define one tableau system for one logic rather than a particular one for every possible input, we have to include patterns for all possible node labels in these rules and clash triggers. For a particular input $\Gamma$, these sets are mapped to the appropriate subsets, which is necessary to ensure decidability in ExpTime.

Formally, a tableau system $S$ is a tuple $(\mathsf{NLE}, \mathsf{GME}, \mathsf{EL}, \cdot^S, k, \mathcal{R}, \mathcal{C})$, where $\mathsf{NLE}$ is the set of all possible node label elements, $\mathsf{GME}$ is the set of global memory elements, $\mathsf{EL}$ is the set of edge labels, $k$ is the maximum pattern depth, i.e. the number of edges on a longest path, and $\mathcal{R}$ and $\mathcal{C}$ are the sets of rules and clash triggers as defined above. The function $\cdot^S$ maps an input $\Gamma$ to the tuple $\Gamma^S = (\mathsf{nle}, \mathsf{gme}, \mathsf{el}, \mathsf{ini})$, where $\mathsf{nle}$, $\mathsf{gme}$ and $\mathsf{el}$ are finite subsets of the corresponding sets in $S$, and $\mathsf{ini} \subseteq \wp(\mathsf{nle}) \times \wp(\mathsf{gme})$ describes the possible initial states for $\Gamma$ ($\wp$ denotes power set).

In order to obtain the results of our framework (ExpTime automata algorithm and a terminating TA) from a tableau system $S$, it has to satisfy three conditions: ExpTime-*admissibility, soundness and p-completeness*, which will be explained in the following paragraphs. Since the formal definitions require a rather complex notation, we will only explain the intuition behind these conditions here and refer the reader to [1] for the details of the definitions.

[1] Usually, this property is called "completeness"; we use the word "saturated" to avoid confusion with the notion of completeness of the decision procedure.

ExpTime**-admissibility.** In order to be *admissible*, tableau systems have to satisfy four conditions:

1. Rules may never remove anything from a pattern, and they must add information (nodes, labels or global memory elements).
2. If $P_s = (V_s, E_s, n_s, \ell_s)$ is a saturated pattern and $P = (V, E, n, \ell)$ is a non-saturated sub-pattern of $P_s$ (i.e. $V \subseteq V_s$ and, for all $v \in V$, $n(v) \subseteq n_s(v)$), then every applicable rule can be applied to $P$ in such a way that the resulting pattern $P'$ is a sub-pattern of $P_s$.
3. Rules may only add elements (to nodes, edges or the global memory) which appear in the subset $\mathsf{nle}$, $\mathsf{el}$ or $\mathsf{gme}$ for the corresponding input $\Gamma$.
4. If $P$ is a clash-trigger, then all super-patterns of $P$ are also clash-triggers.

Conditions 1 and 2 are required to prove termination. However, not all existing TAs satisfy these conditions, e.g. sometimes there are rules which merge two nodes, e.g. the $\leqslant$-rule for $\mathcal{SHIQ}$ in [5], or create more nodes than necessary for a saturated pattern, e.g. the usual definition of the $\exists$-rule. However, for the logics we considered it was possible to reformulate these rules in an admissible way. For example, we can define the rules for $\exists$- and $\geqslant$-formulas non-deterministically (see below).

For ExpTime-*admissibility*, we require in addition to admissibility that the sets $\mathsf{nle}_S$, $\mathsf{el}_S$, $\mathsf{gme}_S$ and $\mathsf{ini}_S$ and the size of their elements are polynomial in the size of $\Gamma$ and can be computed in time exponential in the size of $\Gamma$, and that it can be decided in exponential time whether a rule or a clash trigger is applicable to a pattern.

**Soundness and $p$-completeness.** For a tableau system $S$, it must be shown that if there exists a clash-free and saturated completion tree, then there exists a model (*soundness*), and conversely, that there exists a polynomial $p$ such that for every satisfiable input $\Gamma$, there exists a clash-free and saturated completion-tree whose out-degree is bounded by $p(|\Gamma|)$ ($p$-*completeness*). Here, we distinguish between an $S$-tree *compatible with* $\Gamma$, a tree which is labelled in accordance with $\Gamma^S$, from an $S$-tree *for* $\Gamma$, a tree which can be constructed from an initial tree by rule application. If a saturated and complete $S$-tree compatible with $\Gamma$ exists, then the existence of a saturated and complete $S$-tree for $\Gamma$ follows from the framework (essentially from the admissibility condition). Thus in the proofs, we will only show the existence of an S-tree $((V, E, n, \ell), \mu)$ *compatible* with $\Gamma$, which is defined as follows:

- $\mu \subseteq \wp(\mathsf{gme}_S(\Gamma))$ and $n(x) \subseteq \wp(\mathsf{nle}_S(\Gamma))$ for each $x \in V$;
- $\ell(x, y) \in \mathsf{el}_S(\Gamma)$ for each $(x, y) \in E$;
- there exists $(\Lambda, \nu) \in \mathsf{ini}_S(\Gamma)$ such that $\nu \subseteq \mu$ and $\Lambda \subseteq n(v_0)$ for the root node $v_0$;
- the out-degree of $T$ is bounded by $p(|\Gamma|)$ for a polynomial $p$.

# 3 The Description Logics $\mathcal{SHIO}$ and $\mathcal{SHIQ}^V$

Both $\mathcal{SHIO}$ and $\mathcal{SHIQ}$ are extensions of $\mathcal{SHI}$ (called $\mathcal{ALCHI}_{R^+}$ in [3]), which provides for transitive and inverse roles and role hierarchies. In addition to the $\mathcal{SHI}$ constructs, $\mathcal{SHIO}$ allows for *nominals*, i.e. concepts which have to be interpreted by singleton sets. This makes it possible to express that some concept can only have one instance (e.g. $\mathsf{God}$), or to give names to individuals (e.g. $\mathsf{Rome}$ or $\mathsf{John}$) and use these names in concept definitions ($\mathsf{Roman}$ or $\mathsf{Friend\_of\_John}$). The logic $\mathcal{SHIQ}$ allows for *qualifying number restrictions (QNR)* as described in [5]. For this paper, we introduce the syntactic variant $\mathcal{SHIQ}^V$, which does not include universal and

existential quantification and thus requires fewer rules. For both logics, the presence of transitive roles together with role hierarchies makes it possible to internalise general concept inclusion axioms [5], thus we do not include them in our syntax.

**Definition 1 ($\mathcal{SHIO}$ and $\mathcal{SHIQ}^V$ syntax.)** Let $\mathsf{CON}$ be a set of concept names, $\mathsf{ROL}$ be a set of role names, the set of nominal names $\mathsf{NOM} \subseteq \mathsf{CON}$, and the set of transitive role names $\mathsf{TRA} \subseteq \mathsf{ROL}$. If $r$ is a role name, then both $r$ and $r^-$, the inverse of $r$, are roles. To avoid multiple inverse operators as in $r^{--}$, we use the notation $\overline{r}$, with the meaning $r^-$, if $r$ is a role name, and $s$, if $r$ is an inverse role $s^-$. If $r$ and $s$ are roles, then $r \sqsubseteq s$ is a *role inclusion axiom*. An *RBox* is a finite set of role inclusion axioms. For an RBox $B$, we define the *role hierarchy* $B^+$ as $B^+ = B \cup \{\overline{r} \sqsubseteq \overline{s} \mid r \sqsubseteq s \in B\}$, and by $\sqsubseteq^*_B$ we denote the reflexive-transitive closure of $\sqsubseteq$ on $B^+$. A role $r$ is called *simple* w.r.t. an RBox $R$ if there exists no role $s \in \mathsf{TRA}$ with $s \sqsubseteq^*_B r$ or $\overline{s} \sqsubseteq^*_B r$.

The set of $\mathcal{SHIO}$ concepts is inductively defined as follows: every concept name is a concept; and if $C$ and $D$ are concepts and $r$ is a role, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall r.C$, and $\exists r.C$ are also concepts.

The set of $\mathcal{SHIQ}^V$ concepts is inductively defined as for $\mathcal{SHIO}$, with the restriction that the set $\mathsf{NOM}$ is empty and the quantifiers $\exists$ and $\forall$ are not allowed. In addition, $\mathcal{SHIQ}^V$ provides the following constructors: if $C$ is a concept, $m$ is a non-negative integer and $r$ is a *simple* role, then $(\leqslant m\ r\ C)$ and $(\geqslant m\ r\ C)$ are $\mathcal{SHIQ}^V$ roles. If $r$ is an arbitrary role, then $(\leqslant 0\ r\ C)$ and $(\geqslant 1\ r\ C)$ are also $\mathcal{SHIQ}^V$ concepts.

The concepts $\forall r.C$ and $\exists r.C$ can be expressed in $\mathcal{SHIQ}^V$ by $(\leqslant 0\ r\ \neg C)$ and $(\geqslant 1\ r\ C)$, respectively. Thus, our syntax allows for non-simple roles in QNRs, if they are equivalent to an $\forall$ or $\exists$ formula, but not in the general case.

**Definition 2 ($\mathcal{SHIO}$ and $\mathcal{SHIQ}^V$ semantics.)** An *interpretation* of a concept $C$ w.r.t. an RBox $B$ is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set of individuals and $\cdot^\mathcal{I}$ maps every concept name $C$ to a set $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and every role name $r$ to a set $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. For all $O \in \mathsf{NOM}$, it holds that $\#O^\mathcal{I} = 1$, where $\#S$ denotes the cardinality of a set $S$. For all $t \in \mathsf{TRA}$, it holds that $t^\mathcal{I} = (t^\mathcal{I})^+$, where $\cdot^+$ denotes the transitive closure of a relation $t$. Complex roles and concepts are interpreted as follows:

- $(r^-)^\mathcal{I} = \{(x,y) \mid (y,x) \in r^\mathcal{I}\}$,
- $(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$, $\quad (C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$, $\quad (C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$,
- $(\exists r.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \text{there is an } e \in \Delta^\mathcal{I} \text{ with } (d,e) \in r^\mathcal{I} \text{ and } e \in C^\mathcal{I}\}$,
- $(\forall r.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \text{for all } e \in \Delta^\mathcal{I}, \text{ if } (d,e) \in r^\mathcal{I}, \text{ then } e \in C^\mathcal{I}\}$,
- $(\leqslant m\ r\ C)^\mathcal{I} = \big\{x \mid \sharp\{(x,y) \in r^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \leq m\big\}$,
- $(\geqslant m\ r\ C)^\mathcal{I} = \big\{x \mid \sharp\{(x,y) \in r^\mathcal{I} \text{ and } y \in C^\mathcal{I}\} \geq m\big\}$.

An interpretation $\mathcal{I}$ is a *model* for an RBox $B$ if, for all $r \sqsubseteq s \in B$, it holds that $r^\mathcal{I} \subseteq s^\mathcal{I}$. A *model* for $C$ w.r.t. $B$ is a model for $B$ where $C^\mathcal{I}$ is a nonempty set. If such a model exists, we say that $C$ is *satisfiable* w.r.t. $B$.

## 4 The Tableau Systems $S_\mathcal{O}$ and $S_\mathcal{Q}$

Before defining the tableau systems, we fix some notation. Firstly, in an S-pattern $P = ((V, E, n, \ell), \mu)$ with $\{m, n\} \subseteq V$, we call $m$ an *r-neighbour* of $n$ if $\ell(n,m) = r$ or

$\ell(m,n) = \overline{r}$. Secondly, to capture roles which are implicitly declared to be transitive (e.g. $\overline{r}$ if $r \in \mathsf{TRA}$), we use, for an RBox $B$, the predicate $\mathsf{Trans}_B$, and define that for a role $r$, $\mathsf{Trans}_B(r)$ is true iff there exists a role $s$ such that $s \in \mathsf{TRA}, s' \sqsubseteq^*_B r$ and $r \sqsubseteq^*_B s''$ for some $s', s'' \in \{s, s^-\}$.

For the sake of simplicity, we only deal with concepts in *negation normal form (NNF)*, i.e. where negation appears only directly before concept names. Every concept can be transformed into an equivalent one in NNF in linear time using the duality of $\wedge$ to $\vee$, $\exists$ to $\forall$, and $\geqslant$ to $\leqslant$. By $\sim C$ we denote the NNF of $\neg C$. The *closure* $\mathsf{clos}(C,B)$ of a concept term $C$ and an RBox $B$ is defined as follows: $C \in \mathsf{clos}(C,B)$; if $\neg D \in \mathsf{clos}(C,B)$, then $D \in \mathsf{clos}(C,B)$; if $D \sqcap E$ or $D \sqcup E \in \mathsf{clos}(C,B)$, then $\{D,E\} \subseteq \mathsf{clos}(C,B)$; if $\exists r.D \in \mathsf{clos}(C,B)$, then $D \in \mathsf{clos}(C,B)$; and if $\forall r.D \in \mathsf{clos}(C,B)$ and the role $s$ appears in $C$ or $B$, then $\{D, \forall s.D, \forall \overline{s}.D\} \subseteq \mathsf{clos}(C,B)$.[2] For QNR, we need the following addition: if $(\leqslant m\ r\ D)$ or $(\geqslant m\ r\ D) \in \mathsf{clos}(C,B)$, then $\{D, \sim D\} \subseteq \mathsf{clos}(C,B)$, and if $(\leqslant 0\ r\ D) \in \mathsf{clos}(C,B)$ and the role $s$ appears in $C$ or $B$, then $\{(\leqslant 0\ s\ D), (\leqslant 0\ \overline{s}\ D)\} \subseteq \mathsf{clos}(C,B)$.

We can now define a TS for $\mathcal{SHIO}$, $S_\mathcal{O} = (\mathsf{NLE}_\mathcal{O}, \mathsf{GME}_\mathcal{O}, \mathsf{EL}_\mathcal{O}, 1, \cdot^{S_\mathcal{O}}, \mathcal{R}_\mathcal{O}, \mathcal{C}_\mathcal{O})$. Here, we use the global memory for three purposes: for transitive roles, role inclusion axioms, and for information about concepts appearing in a node label together with a nominal.

- $\mathsf{NLE}_\mathcal{O}$ is the set of all $\mathcal{SHIO}$ concepts,
- $\mathsf{GME}_\mathcal{O} = \{(O,C) \mid O \in \mathsf{NOM}$ and $C \in \mathsf{NLE}_\mathcal{O}\} \cup$ $\{\mathsf{Trans}(r) \mid r$ is a role$\} \cup \{r \sqsubseteq^* s \mid r$ and $s$ are roles$\}$,
- $\mathsf{EL}_\mathcal{O}$ is the set of all $\mathcal{SHIO}$ roles, and
- for an input $\Gamma = (C,B)$, where $C$ is a concept and $B$ is an RBox, the function $\cdot^{S_\mathcal{O}}$ maps $\Gamma$ to a tuple $\Gamma^{S_\mathcal{O}} = (\mathsf{nle}_\Gamma, \mathsf{gme}_\Gamma, \mathsf{el}_\Gamma, \mathsf{ini}_\Gamma)$ with
  - $\mathsf{nle}_\Gamma = \mathsf{clos}(C,B)$,
  - $\mathsf{el}_\Gamma = \{r \mid r$ or $\overline{r}$ appears in $C$ or $B\}$,
  - $\mathsf{gme}_\Gamma = \{(O,D) \mid O \in \mathsf{NOM} \cap \mathsf{clos}(C,B)$ and $D \in \mathsf{clos}(C,B)\} \cup$ $\{\mathsf{Trans}(r) \mid r \in \mathsf{el}_\Gamma\} \cup \{r \sqsubseteq^* s \mid \{r,s\} \subseteq \mathsf{el}_\Gamma\}$, and
  - $\mathsf{ini}_\Gamma = \{(\{C\}, \{\mathsf{Trans}(r) \mid \mathsf{Trans}_B(r)$ holds$\} \cup \{r \sqsubseteq^* s \mid r \sqsubseteq^*_B s$ holds$\})\}$.

The set of rules $\mathcal{R}_\mathcal{O}$ is defined in Figure 1. For each pattern $P = (t, \mu)$, where $t = (V, E, n, \ell)$ has $v_0$ as root and depth at most 1, $\mathcal{R}(P)$ contains the described sets. Most of these rules correspond directly to the "standard" rules known from DL tableaus, with the exception of R$\exists$, which in our framework is non-deterministic. The reason for this is that with a deterministic rule which simply adds a new son node, this TS would violate condition 2 of admissibility (see Section 2). In an implementation, a deterministic rule would be preferable due to efficiency considerations, since the creation of duplicate nodes does not compromise completeness of the decision procedure. Finally, the set $\mathcal{C}_\mathcal{O}$ of clash patterns contains all patterns $((V,E,n,\ell),\mu)$ of depth 0 with node $v_0$ such that $\{D, \neg D\} \subseteq n(v_0)$ for some concept $D \in \mathsf{clos}(C,R)$. This completes $S_\mathcal{O}$, and we can obtain our first result:

**Lemma 3** The TS $S_\mathcal{O}$ is admissible, sound and $q$-complete for $\mathcal{SHIO}$ satisfiability, where $q = (x \mapsto x^2)$.

***Proof.*** Since admissibility of the tableau systems is easy to see, we prove only soundness and $p$-completeness.

[2] The slightly unusual definition for the $\forall$ quantifier is motivated by the $\forall_+$-rule (see below), which in turn is necessary to capture transitive sub-roles of non-transitive roles.

**R⊓** If $C \sqcap D \in n(v)$ for a node $v \in V$ and $\{C, D\} \not\subseteq n(v)$, then $\mathcal{R}(P)$ contains $\{((V, E, n', \ell), \mu)\}$, where $n'(x) = n(x)$ for all $x \neq v$ and $n'(v) = n(v) \cup \{C, D\}$.

**R⊔** If $C \sqcup D \in n(v)$ and $\{C, D\} \cap n(v) = \emptyset$, then $\mathcal{R}(P)$ contains $\{((V, E, n', \ell), \mu), ((V, E, n'', \ell), \mu)\}$, where $n'(x) = n''(x) = n(x)$ for all $x \neq v$, $n'(v) = n(v) \cup \{C\}$ and $n''(v) = n(v) \cup \{D\}$.

**R∃** If $\exists r.C \in n(v_0)$, $v_1, \ldots, v_m$ are all the sons of $v_0$ with $\ell(v_0, v_i) = r$, and $C \notin n(v_i)$ for all $i, 1 \leq i \leq m$, then $\mathcal{R}(P)$ contains the set $\{P_0, P_1, \ldots, P_m\}$ with

- $P_0 = ((V_0, E_0, n_0, \ell_0), \mu)$, where $v' \notin V$, $V_0 = V \cup \{v'\}, E_0 = E \cup \{(v_0, v')\}, n_0 = n \cup \{v' \mapsto \{C\}\}, \ell_0 = \ell \cup \{(v_0, v') \mapsto r\}$.
- for all $i, 1 \leq i \leq m, P_i = ((V, E, n_i, \ell), \mu)$, where $n_i(x) = n(x)$ for all $x \neq v_i$ and $n_i(v_i) = n(v_i) \cup \{C\}$.

**R∀** If $\forall r.C \in n(v)$ for some $v \in V$, $v'$ is an $s$-neighbour of $v$ with $C \notin n(v')$ and $s \sqsubseteq^* r \in \mu$, then $\mathcal{R}(P)$ contains $\{((V, E, n', \ell), \mu)\}$ with $n'(x) = n(x)$ for $x \neq v'$ and $n'(v') = n(v') \cup \{C\}$.

**R∀$_+$** If $\forall r.C \in n(v)$, $\{\mathsf{Trans}(s), s \sqsubseteq^* r, q \sqsubseteq^* s\} \subseteq \mu$ and $v'$ is an $q$-neighbour of $v$ with $\forall s.C \notin n(v')$ , then $\mathcal{R}(P)$ contains $\{((V, E, n', \ell), \mu)\}$ with $n'(x) = n(x)$ for $x \neq v'$ and $n'(v') = n(v') \cup \{\forall s.C\}$.

**R↑** If $\{O, C\} \subseteq n(v)$ for some $O \in \mathsf{NOM}$ and $(O, C) \notin \mu$, then $\mathcal{R}(P)$ contains $\{((V, E, n, \ell), \mu')\}$, where $\mu' = \mu \cup \{(O, C)\}$.

**R↓** If $O \in n(v)$ for an $O \in \mathsf{NOM}, (O, C) \in \mu$ and $C \notin n(v)$, then $\mathcal{R}(P)$ contains $\{((V, E, n', \ell), \mu)\}$, where $n'(x) = n(x)$ for $x \neq v$ and $n'(v) = n(v) \cup \{C\}$.

Figure 1: Tableau rules for $\mathcal{SHIO}$.

**Soundness.** From a saturated and clash-free S-tree $(t, \mu)$ with $t = (V, E, n, \ell)$, we generate a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as follows: $\Delta^{\mathcal{I}} = \{d_O \mid O \in \mathsf{NOM} \cap \mathsf{clos}(C, B)\} \cup \{d_v \mid v \in V \text{ and } n(v) \cap \mathsf{NOM} = \emptyset\}$, i.e. we have one individual for each nominal name and one individual for every tree node that is not labelled with a nominal. A concept name $C$ is interpreted as follows: for every $O \in \mathsf{NOM} \cap \mathsf{clos}(C, B)$, $d_O \in C^{\mathcal{I}}$ iff there is a node $v \in V$ with $\{O, C\} \subseteq n(v)$. Since R↑ and R↓ are not applicable, all nodes whose labels contain the same nominal symbol have exactly the same label, and thus $\cdot^{\mathcal{I}}$ is well-defined. For all other individuals, $d_v \in C^{\mathcal{I}}$ iff $C \in n(v)$. For a role $r$, $r^{\mathcal{I}}$ is the smallest set satisfying the following conditions: if $\ell(v, w) = r$ or $\ell(w, v) = \overline{r}$, then $(d_v, d_w) \in r^{\mathcal{I}}$; if $s \sqsubseteq^* r \in \mu$, then $s^{\mathcal{I}} \subseteq r^{\mathcal{I}}$; if $\mathsf{Trans}(r) \in \mu$, then $r^{\mathcal{I}}$ is closed under transitivity.

We will now show by induction that complex concepts are interpreted correctly. By definition, all individuals belong to the interpretation of the concept names in their labels, and the interpretation of a nominal contains exactly one element. From our construction, it follows directly that the role hierarchy is respected and transitive roles are interpreted correctly. For a conjunct $C \sqcap D$ (disjunct $C \sqcup D$) in a node label $n(v)$, since R⊓ (R⊔) is not applicable, it follows that $C$ and $D$ ($C$ or $D$) are contained in $n(v)$, and by induction, $d_v$ is contained in $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ ($C^{\mathcal{I}} \cup D^{\mathcal{I}}$).

If $\exists r.C \in n(v)$, we assume w.l.o.g. that $r$ is a role name (if it is an inverse role, the argument is analogous). Since R∃ is not applicable, there exists an $r$-son $w$ of $v$ with $C \in n(w)$. By construction, $(d_v, d_w) \in r^{\mathcal{I}}$ and $d_w \in C^{\mathcal{I}}$. If $\forall r.C \in n(v)$, we again assume that $r$ is a role name. There are two possible reasons why $(d_v, d_w)$ can be contained in $r^{\mathcal{I}}$: firstly, if $w$ is an $s$-neighbour of $v$ for some $s$ with $s \sqsubseteq^* r \in \mu$. In

this case, it follows that $C \in n(w)$ because R$\forall$ is not applicable, and thus $d_w \in C^{\mathcal{I}}$. Secondly, if there exist roles $s, s_1, \ldots, s_k$ such that $\{\mathsf{Trans}(s), s \sqsubseteq^* r, s_i \sqsubseteq^* s\} \subseteq \mu$ for all $i \in \{1, \ldots, k\}$ and there is an $s_i$-chain from $v$ to $w$, i.e. a sequence of nodes $v_1, v_2, \ldots, v_n$ such that, for all edges $e \in \{(v, v_1), (v_1, v_2), \ldots, (v_n, w)\}$, it holds that $e \in E$ and $\ell(e) = s_i$ for some $i$. In this case, since R$\forall_+$ is not applicable, all nodes $v_1, \ldots, v_k$ are labelled with $\forall s.C$ and, since R$\forall$ is not applicable to $v_k$, $n(w)$ contains $C$. By induction, it follows that $d_v \in (\forall r.C)^{\mathcal{I}}$.

**Completeness.** We have to show that if there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for an input $\Gamma = (C, B)$, then there also exists a clash-free and saturated S-tree $(t, \mu)$ with $t = (V, E, n, \ell)$ for $\Gamma$, whose width is at most quadratic in $|\Gamma|$. We will create $(t, \mu)$ by unravelling $\mathcal{I}$: firstly, we add the appropriate transitivity axioms ($\mathsf{Trans}(r)$ if $\mathsf{Trans}_B(r)$ holds) and role inclusion axioms ($r \sqsubseteq^* s$ if $r \sqsubseteq^*_B s$ holds) to $\mu$. The tree $t$ is inductively defined as follows: since $\mathcal{I} \models \Gamma$, there is an individual $d_0$ in $\Delta^{\mathcal{I}}$ which satisfies $C$. We start with $V = \{v_0\}$ and define $n(v_0)$ as the set of all concepts in $\mathsf{clos}(C, B)$ which $d_0$ satisfies. We define a function $\pi : V \to \Delta^{\mathcal{I}}$ and set $\pi(v_0) = d_0$.

Then we iterate, for every node $v$, the following procedure: for every existential formula $\exists r.D \in n(v)$ we choose a witness individual $d \in \Delta^{\mathcal{I}}$ with $d \in D^{\mathcal{I}}$ and $(\pi(v), d) \in r^{\mathcal{I}}$ (such a witness exists by definition of $n(v)$). We create a new node $w$ with $\pi(w) = d$, $(v, w) \in E$ and $\ell(v, w) = r$. Again, we label $w$ with the appropriate concepts in $\mathsf{clos}(C, B)$ and then continue the iteration. For every nominal concept $O$, we add to $\mu$ the pair $(O, D)$ for every concept $D \in \mathsf{clos}(C)$ which the unique element $d_O$ of $O^{\mathcal{I}}$ satisfies.

It is easy to see that $(t, \mu)$ is compatible with $\Gamma$ and clash-free. We will now show that it is also saturated: from the definition of $\mathsf{clos}$, it follows that R$\sqcap$ and R$\sqcup$ are not applicable. If a node label $n(v)$ contains a concept $\exists r.D$, then by construction of $t$, there is an $r$-successor of $v$ which is labelled with $D$. Likewise, if $\forall r.D \in n(v)$, all $r$-neighbours of $v$ are labelled with $D$. If $\forall r.D \in n(v)$, $\mu$ contains $s \sqsubseteq^* r$, $q \sqsubseteq^* s$ and $\mathsf{Trans}(s)$, and there is an $q$-neighbour $w$ of $v$, then, since $\mathcal{I}$ is a model, $(\pi(v), \pi(w)) \in s^{\mathcal{I}}$ and, since $s^{\mathcal{I}}$ is transitive, for every node $u$ with $(\pi(w), \pi(u)) \in s^{\mathcal{I}}$, it also holds that $(\pi(v), \pi(u)) \in s^{\mathcal{I}}$, and therefore $\pi(u) \in D^{\mathcal{I}}$. Thus, $\pi(w) \models \forall r.D$ and, since $s \sqsubseteq^*_B r$, $v(w)$ contains $\forall s.D$, which means that R$\forall_+$ is not applicable. Finally, since every node $n$ with $\pi(n) = d_O$ for a nominal $O$ is labelled with exactly those concepts for which $\mu$ contains $(O, C)$, R$\uparrow$ and R$\downarrow$ are not applicable.

The width of the S-tree is quadratic in the length of $\Gamma$, because we create for every node at most one successor for every existential formula in $\mathsf{clos}(\Gamma)$ and the number of such formulas is bounded by the product of the number of roles appearing in $C$ or $B$ and the number of existential subformulas of $C$. ∎

From this, we can derive that $\mathcal{SHIO}$ satisfiability is decidable through a tableau algorithm, and we know that for the blocking condition, equality blocking suffices, i.e. we do not need pair-wise blocking as e.g. for $\mathcal{SHIQ}$ [5], since we use only patterns of depth at most 1. We can also derive a complexity result:

**Theorem 4** Satisfiability for $\mathcal{SHIO}$ concepts w.r.t. RBoxes is ExpTime-complete.

***Proof.*** It is easy to see that $S$ is ExpTime-admissible: e.g. the size of $\mathsf{nle}_{\mathcal{S}}(\Gamma)$ and $\mathsf{gme}_{\mathcal{S}}(\Gamma)$ is quadratic in the size of the input. Soundness and completeness have been shown above. ExpTime-hardness follows from the fact that $\mathcal{SHIO}$ is an extension of $\mathcal{ALC}$ with TBoxes, for which satisfiability is known to be ExpTime-hard [7]. ∎

**R⊓/R⊔** See $\mathcal{R}_\mathcal{O}$.

**RC** If $(\gtrless\ m\ r\ C) \in n(v)$ (where $\gtrless$ is a placeholder for $\geqslant$ or $\leqslant$) for some $m$ and a node $v \in V$ and $\{C, \sim C\} \cap n(w) = \emptyset$ for an $r$-neighbour $w$ of $v$, then $\mathcal{R}$ contains the set $\{((V, E, n', \ell), \mu),\ ((V, E, n'', \ell), \mu)\}$ with $n'(x) = n''(x) = n(x)$ for all $x \in V \setminus \{w\}$ and $n'(w) = n(w) \cup \{C\}$ and $n''(w) = n(w) \cup \{\sim C\}$.

**R∀$_+$** If $(\leqslant 0\ r\ C) \in n(v)$ for a node $v \in V$ and there is a role $s$ with $\{\mathsf{Trans}(s),\ q \sqsubseteq^* s,\ s \sqsubseteq^* r\} \subseteq \mu$ and an $q$-neighbour $w$ of $v$ with $(\leqslant 0\ s\ C) \notin n(w)$, then $\mathcal{R}$ contains $\{((V, E, n', \ell), \mu)\}$ with $n'(x) = n(x)$ for $x \neq w$ and $n'(w) = n(w) \cup \{(\leqslant 0\ s\ C)\}$.

**R$\geqslant$** If $P$ is a pattern of depth 2 and $(\geqslant m\ r\ C) \in n(w)$ for one successor $w$ of $v_0$ and there are less than $m$ $s$-neighbours of $w$ with $s \sqsubseteq^* r \in \mu$ and $C \in n(u_i)$, then $\mathcal{R}$ contains the set $\{P_0, P_1 \dots P_n\}$, where $u_1 \dots u_n$ are the $s$-neighbours of $w$ with $s \sqsubseteq^* r \in \mu$ and $C \notin n(u_i)$ and

- $P_0 = ((V_0, E_0, n_0, \ell_0), \mu)$ with $u_0 \notin V$ ,$V_0 = V \cup \{u_0\}$, $E_0 = E \cup \{(w, u_0)\}$, $n_0(x) = n(x)$ for all $x \in V$ and $n_0(u_0) = \{C\}$ and $\ell_0 = \ell \cup \{(w, u_0) \mapsto s\}$.
- For $1 \leq i \leq n$, $P_i = ((V, E, n_i, \ell), \mu)$ with $n_i(x) = n(x)$ for all $x \in V \setminus \{u_i\}$ and $n_i(u_i) = n(u_i) \cup \{C\}$.

**R$\geqslant_{\mathsf{ROOT}}$** If $\{(\geqslant m\ r\ C), \mathsf{ROOT}\} \in n(v_0)$ of the root node $v_0$ and there are less than $m$ $s$-successors of $v_0$ with $s \sqsubseteq^* r \in \mu$ and $C \in n(u_i)$, then $\mathcal{R}$ contains the set $\{P_0, P_1 \dots P_n\}$, where $u_1 \dots u_n$ are the $s$-successors of $v_0$ with $s \sqsubseteq^* r \in \mu$ and $C \notin n(u_i)$ and $P_0, \dots, P_n$ are defined as for R$\geqslant$.

Figure 2: Tableau Rules for $\mathcal{SHIQ}^V$

For $\mathcal{SHIQ}^V$, we need a TS with quite different properties: to handle QNR in the presence of inverse roles correctly, we need patterns of size 2. However, this makes a special treatment for the root node necessary, since it does not have a predecessor. Thus, we need an additional concept name $\mathsf{ROOT}$ and a special $\geqslant$-rule for the root node. Moreover, in contrast to the algorithm in [5], we do not have a $\leqslant$-rule, but a nondeterministic $\geqslant$-rule, which recycles neighbour nodes if necessary.

The TS $S_\mathcal{Q} = (\mathsf{NLE}_\mathcal{Q}, \mathsf{GME}_\mathcal{Q}, \mathsf{EL}_\mathcal{Q}, 2, \cdot^{S_\mathcal{Q}}, \mathcal{R}_\mathcal{Q}, \mathcal{C}_\mathcal{Q})$ is defined as $S_\mathcal{O}$, with the exception that $\mathsf{NLE}_\mathcal{Q}$ contains the additional element $\mathsf{ROOT}$, $\mathsf{GME}_\mathcal{Q}$ and $\mathsf{gme}_\Gamma$ do not contain any "nominal elements" $(O, C)$ and $\mathsf{ini}_\Gamma = \{(\{C, \mathsf{ROOT}\}, \{\mathsf{Trans}(r) \mid \mathsf{Trans}_B(r) \text{ holds}\} \cup \{r \sqsubseteq^* s \mid r \sqsubseteq^*_B s \text{ holds}\})\}$. The rules $\mathcal{R}_\mathcal{Q}$ are given in Figure 2. Note that no rule modifies $\mu$ and that R$\geqslant$ applies only to patterns whose depth is exactly 2, whereas the other rules apply to patterns of depth at most 2. Here, we obtain an explanation why double-blocking [5] is needed for $\mathcal{SHIQ}$: the maximum required pattern depth is 2.

We do not need an extra rule for concepts of the form $(\leqslant 0\ s\ D)$ to propagate $\sim D$ to all the appropriate neighbours (analogous to R∀), since this is performed by the rule RC. We also do not have a $\leqslant$-rule, but only a corresponding clash trigger: the set $\mathcal{C}_\mathcal{Q}$ contains all patterns in $\mathcal{C}_\mathcal{O}$ and additionally all patterns of depth at most 2 such that $(\leqslant m\ s\ C) \in n(v)$ with $v \in V$ and there are at least $m+1$ $r$-neighbours of $v$ with $r \sqsubseteq^* s \in \mu$. For the proof of $p$-completeness, we require that the numbers in number restrictions are coded unary, since otherwise the width of a model can be exponential in the size of the input. We can then obtain alternative proofs for the known results of $\mathcal{SHIQ}$ decidability and complexity:

**Lemma 5** If unary coding is used in number restrictions, the TS $S_{\mathcal{Q}}$ is ExpTime-admissible, sound and $q$-complete for $\mathcal{SHIQ}^V$ satisfiability, where $q = (x \mapsto x^2)$.

***Proof.*** The soundness proof is easier than for $S_{\mathcal{O}}$, since a completion tree corresponds directly to a model, and we do not have to "merge" nodes labelled with nominals.

**Soundness.** From a saturated and clash-free S-tree $(t, \mu)$ with $t = (V, E, n, \ell)$, we generate a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as follows: $\Delta^{\mathcal{I}} = \{d_v \mid v \in V\}$. For any concept name $D$ and any role $r$ we have the same interpretation as in the proof of soundness for $\mathcal{SHIO}$. Thus, for $\sqcap$ and $\sqcup$ concepts, the proof is analogous to the one for Lemma 3.

We will now show that the QNR $(\leqslant m\ r\ D)$ and $(\geqslant m\ r\ D)$ are also interpreted correctly. Let $(\geqslant m\ r\ C)$ be a concept in $n(v)$ of a node $v$. Then, since the S-tree is saturated, there exist at least $m$ $r$-neighbours $u_1 \dots u_m$ of $v$ with $C \in n(u_i)$ and hence $d_{u_i} \in C^{\mathcal{I}}$. From our construction, it follows that $(d_v, d_{u_i}) \in r^{\mathcal{I}}$ and thus $\sharp\{d \mid (d_v, d) \in r^{\mathcal{I}} \text{ and } d \in C^{\mathcal{I}}\} \geq m$. If $(\leqslant m\ r\ C) \in n(v)$ for a node $v$ and a simple role $r$, then, since the $S$-tree is clash-free, there exist at most $m$ $r$-neighbours $u_1 \dots u_m$ of $v$ with $C \in n(u_i)$ and hence $d_{u_i} \in C^{\mathcal{I}}$. All other $r$-neighbours $w$ contain the concept $\sim D$ because the rule RC is not applicable. Since $r$ is simple, our construction of $r^{\mathcal{I}}$ does not introduce any further $r^{\mathcal{I}}$-neighbours. Thus, it follows that $\sharp\{d \mid (d_v, d) \in r^{\mathcal{I}} \text{ and } d \in C^{\mathcal{I}}\} \leq m$.

For a concept $(\leqslant 0\ r\ C) \in n(v)$ and a non-simple role $r$, the proof is similar to the one for $\forall$-concepts in $\mathcal{SHIO}$: for roles $s, s_1, \dots, s_n$ with $s \in \mathsf{TRA}$ and $s_i \sqsubseteq^* s \sqsubseteq^* r$, it follows from saturatedness of the S-tree that $(\leqslant 0\ r\ C) \in n(w)$ for every node $w$ that is reachable from $v$ via an $s_i$-chain, and thus all $r$-neighbours of $w$ are labelled with $\sim C$ since the tree is clash-free and the rule RC is not applicable.

**Completeness.** We have to show that if there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for an input $\Gamma = (C, B)$, then there also exists a clash-free and saturated S-tree $(t, \mu)$ with $t = (V, E, n, \ell)$ for $\Gamma$. We will create $(t, \mu)$ by unravelling $\mathcal{I}$: the global memory $\mu$ is created as in the case of $\mathcal{SHIO}$ by adding all appropriate transitivity and role inclusion axioms. The tree is inductively defined as follows: we start with an individual $d_0 \in C^{\mathcal{I}}$, create a node $v_0$, and a function $\pi$ with $\pi(v_0) = d_0$. Moreover, we define, for this and all further nodes $v$, $n(v)$ as the set of all concepts $D \in \mathsf{clos}(C, B)$ which the individual $\pi(v)$ satisfies. For the root node, we add the marker concept $\mathsf{ROOT}$ to $n(v_0)$.

New nodes are added to the tree if there is a node $v$ and a formula $(\geqslant m\ r\ D) \in n(v)$: if there exist only $k$ $r$-neighbours of $v$ and $k < m$, we choose appropriate individuals $d_{k+1} \dots d_m$, i.e. individuals which are not yet in the range of $\pi$, with $d_i \in D^{\mathcal{I}}$ and $(\pi(v), d_i) \in r^{\mathcal{I}}$. For these nodes, we create $r$-neighbours $u_1 \dots u_m$ of $v$ and set $\pi(u_i) = d_i$. Since $\mathcal{I}$ is a model, it always is possible to find appropriate individuals. From this construction, it follows that R$\geqslant$ and R$\geqslant_{\mathsf{ROOT}}$ are not applicable. The rule RC is not applicable by definition, since every node satisfies either $C$ or $\sim C$ for any concept $C$, and for R$\sqcap$, R$\sqcup$ and R$\forall_+$, saturatedness follows analogous to the proof for $S_{\mathcal{O}}$. Note that the out-degree of the S-tree is bounded by the number of concepts of the form $(\geqslant m\ r\ D) \in \mathsf{clos}(C, B)$ and the highest number $m$ occurring in such a concept.

The resulting S-tree can neither contain a clash trigger with $\{C, \sim C\} \subseteq n(v)$ for a node $v$ and a concept $C$, since $\mathcal{I}$ is a model, nor a clash trigger with a number restriction of the form $D = (\leqslant m\ r\ C) \in n(v)$ of a node $v \in V$ with $\pi(v) = d$, because $d \in D^{\mathcal{I}}$ and we create at most one $r$-neighbour of $v$ for every $r^{\mathcal{I}}$-neighbour of $d$.

It is easy to see that $(t, \mu)$ is compatible with $\Gamma$ and the width is at most quadratic in length of $\Gamma$ since we create only $m$ successors for a formula $(\geqslant\ m\ r\ C) \in \mathsf{clos}(C, B)$, the number $m$ is coded unary and the number of such formulas is quadratic in the length of $\Gamma$. ∎

**Theorem 6** Satisfiability for $\mathcal{SHIQ}^V$ concepts w.r.t. RBoxes is ExpTime-complete.

***Proof.*** The tableau system $S_{\mathcal{Q}}$ is ExpTime-admissible: the size of $\mathsf{nle}_S(\Gamma)$ and of $\mathsf{gme}_S(\Gamma)$ is quadratic in the size of the input. Soundness and $p$-completeness have been shown above. ExpTime-hardness follows as for $S_{\mathcal{O}}$. ∎

# 5 Conclusion and Outlook

We have described the main features of the tableau framework for ExpTime logics and defined tableau systems for the new DL $\mathcal{SHIO}$ and the known DL $\mathcal{SHIQ}$. It turned out that these two logics make use of different features of the tableau framework: to capture nominals, we need the global memory, whereas large patterns are needed to handle QNR properly. From the tableau systems, we can derive automata algorithms deciding satisfiability of $\mathcal{SHIO}/\mathcal{SHIQ}$ concepts w.r.t. RBoxes in ExpTime and terminating tableau algorithms, which are well suited for implementation and include appropriate blocking conditions. We believe that the simplicity of the proofs justifies the additional overhead resulting from the formalisation of the algorithm within the tableau framework.

We aim at extending our framework to cover NExpTime logics, e.g. $\mathcal{ALCQIO}$ [8]. One way of achieving this could consist in replacing looping automata (for which the emptiness problem is in P) with an automata model with an NP-complete emptiness problem, e.g. Rabin automata. However, we do not yet see a way of capturing the $\mathcal{ALCQIO}$-specific problems with a Rabin acceptance condition.

# References


[1] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From tableaux to automata for description logics. *Fundamenta Informaticae*, 57:1–33, 2003.

[2] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69, 2001.

[3] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. Technical Report 98-05, LuFg Theoretical Computer Science, RWTH Aachen, 1998.

[4] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In *Proc. of IJCAI-01*, pages 199–204. Morgan Kaufmann, 2001.

[5] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, number 1705 in LNAI, pages 161–180. Springer-Verlag, 1999.

[6] U. Sattler and M. Y. Vardi. The hybrid $\mu$-calculus. In *IJCAR-01*, volume 2083 of *LNAI*, pages 76–91. Springer-Verlag, 2001.

[7] K. Schild. Terminological cycles and the propositional $\mu$-calculus. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of KR-94*, pages 509–520, Bonn, 1994. Morgan Kaufmann.

[8] S. Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. of Artificial Intelligence Research*, 12:199–217, May 2000.

# Attribute Inversion in Description Logics with Path Functional Dependencies

David Toman and Grant Weddell

School of Computer Science, University of Waterloo
Email: {david,gweddell}@uwaterloo.ca

**Abstract**

We present a coherence condition for a boolean complete description logic with feature inversions and a very general form of uniqueness constraint that enables, among other things, the capture of unary functional dependencies. The condition is sufficiently weak to allow the transfer of relational and emerging object-oriented normalization techniques while still ensuring that the associated logical implication problem remains DEXPTIME-complete.

## 1 Introduction

For many applications, there is considerable incentive to enhance the modelling utility of a description logic (DL) with an ability to capture richer varieties of uniqueness constraints such as keys and functional dependencies [6, 9, 13, 15]. Unfortunately, in combination with role or attribute inversions, the associated logical implication problem quickly becomes undecidable [5]. We present a coherence condition for a boolean complete DL with feature inversions which allows unrestricted use of path functional dependencies [17]. The condition ensures that the associated logical implication problem remains DEXPTIME-complete, but is sufficiently weak to allow the formal specification of arbitrary relational or object-oriented schema, including those that fail to satisfy normalization conditions. This latter observation is important since it enables an incremental development of terminologies that encode schema. One can begin, for example, with a "relational" terminology that fails to satisfy the conditions of Boyce-Codd Normal Form. (Note that the approach used in [5] is not generally capable of handling such anomalous cases.) Standard normalization algorithms and methodology can then employ reasoning services based on our results. Thus, our DL is better equipped to enable the transfer of results in normalization and

emerging object design theory for relational and object-oriented data models [2, 3]. We also show that relaxing this coherence condition leads to undecidability.

### 1.1 Related Work

Our coherence condition derives from a similar condition proposed in [4] to enable the development of a sound and complete axiomatization for an object-oriented data model, which essentially adds inclusion dependencies to an earlier data model [17]. The DL we consider in this paper is a further generalization; thus, our DEXPTIME-completeness result *settles an open problem on the decidability of the implication problem for their model.*

In [5], the authors consider a DL with (relational) functional dependencies together with a general form of keys called *identification constraints*. They show that this dialect is undecidable in the general case, but becomes decidable when unary functional dependencies are disallowed. Our coherency condition serves as an alternative method for regaining decidability.

A form of key dependency with left-hand-side feature paths is considered for a DL coupled with various concrete domains [12]. The authors explore how the complexity of satisfaction is influenced by the selection of a concrete domain together with various syntactic restrictions on the key dependencies themselves. We consider a DL that admits more general kinds of key constraints (and functional dependencies) for which identifying values can be defined on arbitrary domains.

The remainder of the paper is organized as follows. Definitions of the DL dialect $\mathcal{DLFAD}$ and the above-mentioned coherency condition are given next in Section 2. In Section 3, we show that failure to satisfy the coherency condition leads to undecidability of the implication problem for $\mathcal{DLFAD}$. For cases satisfying this condition, we show in Section 4 that the problem is DEXPTIME-complete. To do this, we first consider the problem for $\mathcal{DLFA}$, a fragment of $\mathcal{DLFAD}$ that excludes uniqueness constraints. Although $\mathcal{DLFA}$ is less expressive than, e.g., $\mathcal{DLR}$, we gain the opportunity of presenting encoding schemes for reductions in a more incremental fashion. Our summary comments follow in Section 5.

## 2 Preliminaries

**Definition 1 (Description Logic $\mathcal{DLFAD}$)** *Let $F$ and $C$ be sets of attribute names and primitive concept names, respectively. A* path expression *is defined by the grammar "$\mathsf{Pf} ::= f.\mathsf{Pf} \mid Id$" for $f \in F$. We define derived* concept descriptions *by the grammar on the left-hand-side of Figure 1. A concept description*

| | Syntax | Semantics: Defn of "$(\cdot)^{\mathcal{I}}$" |
|---|---|---|
| $D$ ::= | $C$ | $(C)^{\mathcal{I}} \subseteq \Delta$ |
| | $D_1 \sqcap D_2$ | $(D_1)^{\mathcal{I}} \cap (D_2)^{\mathcal{I}}$ |
| | $\neg D$ | $\Delta \setminus (D)^{\mathcal{I}}$ |
| | $\forall f.D$ | $\{x : (f)^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}\}$ |
| | $D@f$ | $\{(f)^{\mathcal{I}}(x) : x \in (D)^{\mathcal{I}}\}$ |
| $E$ ::= | $D$ | |
| | $D : \mathsf{Pf}_1, ..., \mathsf{Pf}_k \rightarrow \mathsf{Pf}$ | $\{x : \forall\, y \in (D)^{\mathcal{I}}.$ $\bigwedge_{i=1}^{k} (\mathsf{Pf}_i)^{\mathcal{I}}(x) = (\mathsf{Pf}_i)^{\mathcal{I}}(y) \Rightarrow (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\}$ |

Figure 1: Syntax and Semantics of $\mathcal{DLFAD}$.

*obtained by using the final production of this grammar is called a* path functional dependency *(PFD).*

*An* inclusion dependency *$\mathcal{C}$ is an expression of the form $D \sqsubseteq E$. A* terminology *$\mathcal{T}$ consists of a finite set of inclusion dependencies.*

*The* semantics *of expressions is defined with respect to a structure $(\Delta, \cdot^{\mathcal{I}})$, where $\Delta$ is a domain of "objects" and $(.)^{\mathcal{I}}$ an interpretation function that fixes the interpretations of primitive concepts $C$ to be subsets of $\Delta$ and primitive attributes $f$ to be total functions $(f)^{\mathcal{I}} : \Delta \rightarrow \Delta$. The interpretation is extended to path expressions, $(Id)^{\mathcal{I}} = \lambda x.x$, $(f.\,\mathsf{Pf})^{\mathcal{I}} = (\mathsf{Pf})^{\mathcal{I}} \circ (f)^{\mathcal{I}}$ and derived concept descriptions $D$ and $E$ as defined on the right-hand-side of Figure 1.*

*An interpretation* satisfies an inclusion dependency *$D \sqsubseteq E$ if $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$. The* logical implication problem *asks if $\mathcal{T} \models D \sqsubseteq E$ holds; that is, if $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$ for all interpretations that satisfy all constraints in $\mathcal{T}$.*

The coherency condition on which our decidability results depend is defined as follows. (Recall that a similar condition is introduced in [4] to ensure that an axiomatization for their data model is complete.)

**Definition 2 (Coherent Terminology)** *A terminology $\mathcal{T}$ is* coherent *if*

$$\mathcal{T} \models (D@f) \sqcap (E@f) \sqsubseteq (D \sqcap E)@f$$

*holds for all descriptions $D, E$ and attributes $f$.*

Note that we can *syntactically guarantee* that $\mathcal{T}$ is coherent by adding the $(D@f) \sqcap (E@f) \sqsubseteq (D \sqcap E)@f$ assertions to $\mathcal{T}$ (for all descriptions $D, E$ that appear in $\mathcal{T}$).

# 3 Undecidability for General $\mathcal{DLFAD}$ Terminologies

We show a reduction of the unrestricted tiling problem to the $\mathcal{DLFAD}$ implication problem using a construction similar to that presented in [5]. An unrestricted tiling problem $U$ is a triple $(T, H, V)$ where $T$ is a finite set of tile types and $H, V \subseteq T \times T$ two binary relations. A *solution* to $T$ is a mapping $t : \mathbf{N} \times \mathbf{N} \to T$ such that $(t(i,j), t(i+1,j)) \in H$ and $(t(i,j), t(i,j+1)) \in V$ for all $i \in \mathbf{N}$. This problem is $\Pi^0_0$-complete [1, 16]. The first step in the reduction is to establish an *integer grid*. This can be achieved, for example, as follows.

1. Introduce four disjoint concepts, $A$, $B$, $C$ and $D$, denoting cell edges.

$$\mathrm{A} \sqcap \mathrm{B} \sqsubseteq \bot, \quad \mathrm{A} \sqcap \mathrm{C} \sqsubseteq \bot, \quad \ldots, \quad \mathrm{C} \sqcap \mathrm{D} \sqsubseteq \bot$$

2. Grid cells are mapped to concepts X and Y that have four incoming $f$ and $g$ attributes, respectively.

$$\mathrm{X} \sqsubseteq (\mathrm{A}@f) \sqcap (\mathrm{B}@f) \sqcap (\mathrm{C}@f) \sqcap (\mathrm{D}@f),$$
$$\mathrm{Y} \sqsubseteq (\mathrm{A}@g) \sqcap (\mathrm{B}@g) \sqcap (\mathrm{C}@g) \sqcap (\mathrm{D}@g)$$

3. To ensure that squares are formed, add the following.

$$\mathrm{A} \sqsubseteq \mathrm{B} : f \to h, \quad \mathrm{B} \sqsubseteq \mathrm{C} : f \to i, \quad \mathrm{C} \sqsubseteq \mathrm{D} : f \to h, \quad \mathrm{D} \sqsubseteq \mathrm{A} : f \to i,$$
$$\mathrm{A} \sqsubseteq \mathrm{B} : h \to f, \quad \mathrm{B} \sqsubseteq \mathrm{C} : i \to f, \quad \mathrm{C} \sqsubseteq \mathrm{D} : h \to f, \quad \mathrm{D} \sqsubseteq \mathrm{A} : i \to f,$$
$$\mathrm{A} \sqsubseteq \mathrm{B} : g \to i, \quad \mathrm{B} \sqsubseteq \mathrm{C} : g \to h, \quad \mathrm{C} \sqsubseteq \mathrm{D} : g \to i, \quad \mathrm{D} \sqsubseteq \mathrm{A} : g \to h,$$
$$\mathrm{A} \sqsubseteq \mathrm{B} : i \to g, \quad \mathrm{B} \sqsubseteq \mathrm{C} : h \to g, \quad \mathrm{C} \sqsubseteq \mathrm{D} : i \to g, \quad \mathrm{D} \sqsubseteq \mathrm{A} : h \to g$$

4. And to force squares to extend to the right and up, include the following.

$$\mathrm{A} \sqsubseteq \forall g.Y, \quad \mathrm{B} \sqsubseteq \forall g.Y, \quad \mathrm{C} \sqsubseteq \forall f.X, \quad \mathrm{D} \sqsubseteq \forall f.X$$

The *adjacency rules* from the instance $U$ of the tiling problem are defined as follows:

$$A \sqcap \forall f.T_i \sqsubseteq \forall g. \bigsqcup_{(t_i,t_j)\in V} T_j, \qquad C \sqcap \forall g.T_i \sqsubseteq \forall f. \bigsqcup_{(t_i,t_j)\in V} T_j$$
$$B \sqcap \forall f.T_i \sqsubseteq \forall g. \bigsqcup_{(t_i,t_j)\in H} T_j, \qquad D \sqcap \forall g.T_i \sqsubseteq \forall f. \bigsqcup_{(t_i,t_j)\in H} T_j,$$

where $T_i$ corresponds to a tile $t_i \in T$; we assume $T_i \sqcap T_j \sqsubseteq \bot$ for all $i < j$. The above constraints form a terminology $\mathcal{T}_U$ associated with an unrestricted tiling problem $U$.

**Theorem 3** *A tiling problem $U$ admits a solution iff $\mathcal{T}_U \not\models \mathrm{X} \sqcap (\bigsqcup_{t_i \in T} T_i) \sqsubseteq \bot$.*

Thus, the $\mathcal{DLFAD}$ implication problem is undecidable for unrestricted terminologies.

# 4 Coherency implies Decidability

By restricting logical implication problems for $\mathcal{DLFAD}$ to cases in which terminologies are coherent, it becomes possible to apply reductions to satisfiability problems for Ackerman formulae. After we introduce the latter, we begin by defining reductions for the fragment $\mathcal{DLFA}$. An essentially incremental elaboration of these reductions is presented in the final subsection in which we establish our main result: decidability with coherency of the logical implication problem for $\mathcal{DLFAD}$.

**Definition 4 (Monadic Ackerman Formulae)** *Let $P_i$ be monadic predicate symbols and $x, y_i, z_i$ variables. A* monadic first-order formula in the Ackermann class *is a formula of the form $\exists z_1 \ldots \exists z_k \forall x \exists y_1 \ldots \exists y_l . \varphi$ where $\varphi$ is a quantifier-free formula over the symbols $P_i$.*

Every formula with the Ackermann prefix can be converted to *Skolem normal form*: by replacing variables $z_i$ by Skolem constants and $y_i$ by unary Skolem functions not appearing in the original formula. This, together with standard boolean equivalences, yields a finite set of universally-quantified clauses containing at most one variable ($x$). It is known that an Ackerman sentence has a model if and only if it has a Herbrand model; this allows us to use syntactic techniques for model construction. To establish the complexity bounds we use the following result for the satisfiability of Ackermann formulae:

**Proposition 5 ([7])** *The complexity of the satisfiability problem for Ackerman formulae is DEXPTIME-complete.*

## 4.1 Decidability for Coherent $\mathcal{DLFA}$ Terminologies

We construct an Ackerman-class sentence whose satisfiability is equivalent to a given $\mathcal{DLFA}$ implication problem. In the simulation, $\mathcal{DLFA}$'s concept descriptions $D$ are modeled by monadic predicates $P_D(x)$. The function symbols $f$ and $\overline{f}$ are used in sentences to stand for the attribute $f$; the symbol $\overline{f}$ stands for the reverse of the attribute $f$ in cases where an object is in the range of multiple attributes (this situation can be introduced, e.g., by the description $(\mathrm{D}_1 @ f_1) \sqcap (\mathrm{D}_2 @ f_2)$). This arrangement, in the case of coherent terminologies, allows us to represent $\mathcal{DLFA}$ interpretations as Herbrand interpretations (in the extended language). The construction proceeds in two steps. First, the structural properties of $\mathcal{DLFA}$ are encoded using the following assertions.

- Node existence assertions:

$$\forall x.N(x) \leftrightarrow N(f_i(x)) \;\text{ for }\; x \neq \overline{f}_i(y), \quad \forall x.N(\overline{f}_i(x)) \leftrightarrow N(x)$$

- Functionality of attributes and Coherence:

$$\forall x.\neg(N(x) \wedge N(f(\overline{f}(x))), \qquad \forall x.\neg(N(x) \wedge N(\overline{f}(f(x)))$$

- Concept formation assertions for boolean constructors:

$$\forall x.N(x) \rightarrow (P_D(x) \vee P_{\neg D}(x)), \quad \forall x.\neg(P_D(x) \wedge P_{\neg D}(x)),$$
$$\forall x.N(x) \rightarrow (P_{D_1 \sqcap D_2}(x) \leftrightarrow (P_{D_1}(x) \wedge P_{D_2}(x)))$$

- Concept formation assertions for attribute constructors:

$$\begin{array}{rcl}
\forall x.N(x) & \rightarrow & (P_{\forall f_i.D}(x) \leftrightarrow P_D(f_i(x))) \text{ for } x \neq \overline{f}_i(y) \\
\forall x.N(\overline{f}_i(x)) & \rightarrow & (P_{\forall f_i.D}(\overline{f}_i(x)) \leftrightarrow P_D(x)) \\
\forall x.N(x) & \rightarrow & (P_{D@f_i}(x) \rightarrow N(\overline{f}_i(x))) \text{ for } x \neq f_i(y) \\
\forall x.N(x) & \rightarrow & (P_{D@f_i}(x) \leftrightarrow P_D(\overline{f}_i(x))) \text{ for } x \neq f_i(y) \\
\forall x.N(\overline{f}_i(x)) & \rightarrow & (P_{D@f_i}(f_i(x)) \leftrightarrow P_D(x))
\end{array}$$

The collection of the assertions above is denoted $\Pi_{\mathcal{DLFA}}$. This set captures the structural relationships between $\mathcal{DLFA}$ concepts. Although this set is infinite in general, the set of concepts appearing in a particular implication problem, $\mathcal{T} \models \mathcal{C}$, is finite. Hence, one can restrict the set of assertions $\Pi_{\mathcal{DLFA}}$ to a finite subset $\Pi^{\mathcal{T},\mathcal{C}}_{\mathcal{DLFA}}$ that contains only the predicates that define concepts in $\mathcal{T} \cup \{\mathcal{C}\}$. (In the rest of the paper we omit the superscript whenever clear from the context.) To complete the translation of a $\mathcal{DLFA}$ implication problem, what remains is the translation of the inclusion constraints.

**Definition 6** *Let $\mathcal{T}$ and $\mathcal{C} \equiv D \sqsubseteq E$ be a $\mathcal{DLFA}$ terminology and an inclusion constraint, respectively. We define*

- $\Pi_{\mathcal{T}} = \{N(x) \rightarrow (\forall x.P_D(x) \rightarrow P_E(x)) : D \sqsubseteq E \in \mathcal{T}\}$ *and*
- $\Pi_{\mathcal{C}} = \{N(0), P_D(0), P_{\neg E}(0)\}$,

The three clauses $\Pi_{\mathcal{C}}$ represent the skolemized version of $\neg\forall x.P_D(x) \rightarrow P_E(x)$; 0 is the Skolem constant for $x$. As usual, a model "containing" $\Pi_{\mathcal{C}}$ is a counterexample for $\mathcal{C}$. To show the correspondence formally we need the following definition and lemma:

**Definition 7** *An interpretation $(\Delta, (.)^{\mathcal{I}})$ is* coherent *if $(f_i)^{\mathcal{I}}(x) = (f_i)^{\mathcal{I}}(y) \rightarrow x = y$ for all $x, y \in \Delta$ and $f_i$ an attribute name.*

**Lemma 8** *Let $\mathcal{T}$ be a coherent terminology, $\mathcal{C}$ a subsumption constraint, and $\mathcal{I}$ an interpretation such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models \mathcal{C}$. Then there is a coherent interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \mathcal{T}$ and $\mathcal{I}' \not\models \mathcal{C}$.*

**Proof:** (sketch) Consider distinct $x, y \in \Delta_{\mathcal{I}}$ such that (i) $x \in (D_1)^{\mathcal{I}}$, (ii) $y \in (D_2)^{\mathcal{I}}$, and (iii) $(f_i)^{\mathcal{I}}(x) = (f_i)^{\mathcal{I}}(y)$. Then, since $\mathcal{T}$ is coherent, $x \in (D_1 \sqcap D_2)^{\mathcal{I}}$. For, $x \in (D_1 \sqcap \neg D_2)^{\mathcal{I}}$ leads to $(f_i)^{\mathcal{I}}(x) \in ((D_1 \sqcap \neg D_2)@f_i \sqcap D_2@f_i)^{\mathcal{I}}$, a contradiction. Thus, as models of $\mathcal{DLFA}$ have the tree model property, we can remove the farther of $x$ or $y$ and all its descendants, where the distance is measured from the node falsifying $\mathcal{C}$ in $\mathcal{I}$. The resulting interpretation still satisfies $\mathcal{T}$ and falsifies $\mathcal{C}$. Repeating this process yields a coherent interpretation. □

**Theorem 9** *Let $\mathcal{T}$ and $\mathcal{C}$ be a terminology and inclusion dependency in $\mathcal{DLFA}$, respectively. Then $\mathcal{T} \models \mathcal{C} \iff \Pi_{\mathcal{DLFA}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable.*

**Proof:** (sketch) Consider a Herbrand model $\mathcal{M}$ such that $\mathcal{M} \models \Pi_{\mathcal{DLFA}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$. We construct an interpretation $\mathcal{I}_{\mathcal{M}} = (\Delta, (.)^{\mathcal{I}})$ where:

- $\Delta = \{x : \mathcal{M} \models N(x)\}$,
- $(D)^{\mathcal{I}} = \{x : \mathcal{M} \models P_D(x)\}$, and $(f)^{\mathcal{I}} = \{(x, y) : y = f(x) \text{ or } \overline{f}(y) = x\}$.

It is easy to verify (by cases analysis) that $\mathcal{I}_{\mathcal{M}} \models \mathcal{T}$ but $\mathcal{I}_{\mathcal{M}} \not\models \mathcal{C}$.

For the other direction we take any coherent interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models \mathcal{C}$. This interpretation must exist by Lemma 8 whenever $\mathcal{T} \not\models \mathcal{C}$. Let $o \in \Delta$ be an object that falsifies $\mathcal{C}$ in $\mathcal{I}$. We construct a Herbrand universe as the set of all terms that correspond to undirected paths of attributes originating in $o$; we use $\overline{f}_i$ for every attribute $f$ traversed "backward" along this path. Each of these terms, $t_x$, due to the coherence condition, corresponds to exactly one element of $x \in \Delta$. On top of this universe we define a Herbrand model $\mathcal{M}_{\mathcal{I}} = \{P_D(t_x) : x \in (D)^{\mathcal{I}}\} \cup \{N(t_x)\}$ The remainder is verification of $\mathcal{M}_{\mathcal{I}} \models \Pi_{\mathcal{DLFA}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ by cases analysis. □

The translation therefore provides a DEXPTIME decision procedure by appealing to Proposition 5. Completeness follows from DEXPTIME-hardness of the implication problem for the $\{D_1 \sqcap D_2, \forall f.D\}$ fragment [14, 15].

**Corollary 10** *The implication problem for $\mathcal{DLFA}$ is DEXPTIME-complete.*

## 4.2 Decidability for Coherent $\mathcal{DLFAD}$ Terminologies

For each implication problem $\mathcal{T} \models \mathcal{C}$, we define a satisfiability problem $\Pi_{\mathcal{DLFAD}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$. There are two cases to consider depending on $\mathcal{C}$.

**Case 1:** $\mathcal{C}$ is a $\mathcal{DLFA}$ inclusion dependency.

**Lemma 11** *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFAD}$ implication problem in which $\mathcal{T}$ is coherent and for which $\mathcal{C}$ is a $\mathcal{DLFA}$ inclusion dependency. Let $\mathcal{T}'$ be the largest subset of $\mathcal{T}$ that is also a $\mathcal{DLFA}$ terminology. Then $\mathcal{T}' \models \mathcal{C}$ if and only if $\mathcal{T} \models \mathcal{C}$.*

**Proof:** Assume $\mathcal{T}' \not\models \mathcal{C}$. Then by Lemma 8 there must be a coherent interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}'$ but $\mathcal{I} \not\models \mathcal{C}$. However, since $\mathcal{I}$ is coherent, it also satisfies $\mathcal{T}$. The other direction is immediate as $\mathcal{T}' \subseteq \mathcal{T}$. □

Thus, in this case, we can use Theorem 9 to decide the implication problem.

**Case 2:** $\mathcal{C} = D_1 \sqsubseteq D_2 : \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf}$. To falsify such an inclusion dependency, *two* objects (one in $D_1$ and another in $D_2$) that satisfy the preconditions of the dependency but fail to satisfy the conclusion are needed. We therefore construct two copies of the interpretation for the $\mathcal{DLFA}$ constraints in $\mathcal{T}$ in a fashion analogous to [8, 18]. However, as Herbrand terms are essentially the same in the two copies, it is sufficient to distinguish them by renaming the predicate symbols [10]. In addition, we need to model the "rules" of equality and their interaction with concept descriptions. The structural rules for $\mathcal{DLFAD}$ are thus defined as follows:

$$\Pi_{\mathcal{DLFAD}} = \Pi^L_{\mathcal{DLFA}} \cup \Pi^R_{\mathcal{DLFA}} \cup \left\{ \begin{array}{l} \forall x.E(x) \rightarrow E(f_i(x)) \\ \forall x.(N(\overline{f}_i(x) \wedge E(\overline{f}_i(x))) \rightarrow E(x) \\ \forall x.(N(x) \wedge E(x)) \rightarrow (P^L_D(x) \leftrightarrow P^R_D(x)) \end{array} \right\},$$

where $\Pi^L$ is the set of assertions $\Pi$ in which every predicate $P_D$ is renamed to $P^L_D$ (similarly for $\Pi^R$). In addition, we use the following notation: we say that $\overline{\mathsf{Pf}}$ is a *reverse prefix of* $\mathsf{Pf}$ iff $\overline{\mathsf{Pf}}$ is a prefix of $\mathsf{Pf}$ in which each $f_i$ was replaced by $\overline{f}_i$ and the order of the attributes was reversed. We also define

$$\lfloor \mathsf{Pf} \rfloor = \left\{ \begin{array}{ll} \lfloor \mathsf{Pf}_1 \circ \mathsf{Pf}_2 \rfloor & \text{for } \mathsf{Pf} = \mathsf{Pf}_1 \circ \overline{f}_i \circ f_i \circ \mathsf{Pf}_2 \\ \mathsf{Pf} & \text{otherwise} \end{array} \right.$$

We construct a set of assertions for a given terminology $\mathcal{T}$. Let $\mathcal{T}'$ denote the subsumption constraints in $\mathcal{T}$ without PFDs, and $\mathcal{T}'' = \mathcal{T} - \mathcal{T}'$. Then we define

$$\begin{array}{rcl} \Pi_{\mathcal{T}} & = & \Pi^L_{\mathcal{T}'} \cup \Pi^R_{\mathcal{T}'} \\ & \cup & \left\{ \begin{array}{l} N^L(\overline{\mathsf{Pf}}(x)) \wedge N^R(\overline{\mathsf{Pf}}(x)) \wedge \\ ((P^L_{D_1}(\overline{\mathsf{Pf}}(x)) \wedge P^R_{D_2}(\overline{\mathsf{Pf}}(x))) \vee (P^R_{D_1}(\overline{\mathsf{Pf}}(x)) \wedge P^L_{D_2}(\overline{\mathsf{Pf}}(x)))) \wedge \\ \qquad \bigwedge_{i=1}^{k} E(\lfloor \overline{\mathsf{Pf}} \circ \mathsf{Pf}_i \rfloor(x) \rightarrow E(\lfloor \overline{\mathsf{Pf}} \circ \mathsf{Pf}_0 \rfloor(x) \\ \qquad \text{where } \mathrm{D}_1 \sqsubseteq D_2 : \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf}_0 \in \mathcal{T}'' \\ \qquad \text{and } \overline{\mathsf{Pf}} \text{ a reverse prefix of } \mathsf{Pf}_i \end{array} \right\} \end{array}$$

and, for $\mathcal{C} \equiv \mathrm{D}_1 \sqsubseteq D_2 : \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf}_0$ we define

$$\Pi_{\mathcal{C}} = \{N(0), P^L_{D_1}(0), P^R_{D_2}(0), E(\mathsf{Pf}_1(0)), \ldots, E(\mathsf{Pf}_k(0)), \neg E(\mathsf{Pf}_0(0))\}$$

**Theorem 12** *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFAD}$ implication problem in which $\mathcal{C}$ is a PFD. Then $\mathcal{T} \models \mathcal{C}$ if and only if $\Pi_{\mathcal{DLFAD}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable.*

**Proof:** (sketch) The proof proceeds analogously to the proof of Theorem 9 by explicitly constructing a counterexample interpretation from a Herbrand model, and vice versa. The crux lies in observing that, in addition to proper simulation of $\mathcal{DLFA}$ descriptions, a path agreement (i.e., a precondition or a consequent of a PFD) holds in a $\mathcal{DLFAD}$ interpretation if and only if a corresponding $E(t(0))$ atom appears in the Herbrand model—this fact hinges on introducing the $\overline{f}_i$ function symbols and on representing a single PFD by *multiple* formulae. □

# 5 Summary and Future Work

We have defined a coherence condition for a boolean complete description logic with feature inversions and arbitrary path functional dependencies that ensures the associated logical implication problem is DEXPTIME-complete; the problem is undecidable otherwise. This resolves an open issue on decidability of an analogous implication problem in [4].

A natural extension of the description logic presented here allows *regular languages* ($L$) to replace path expressions, yielding the $\forall L.D$, $\exists L.D$, $D@L$, and $D : L \rightarrow L'$ constructors, and developing a decision procedure using the approach in [15]. One of the main applications of such an extension we envision is describing data structures for purposes of query optimization, extending [11] to inductive data types.

Another direction of research considers weaker restrictions on $\mathcal{DLFAD}$ terminologies that still guarantee decidability, e.g., relaxing our *coherence* condition with respect to the unary PFDs actually present in a terminology.

# References


[1] R. Berger. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc.*, 66:1–72, 1966.

[2] Joachim Biskup, Ralf Menzel, Torsten Polle, and Yehoshua Sagiv. Decomposition of Relationships through Pivoting. In *Conceptual Modeling - ER'96*, pages 28–41, 1996.

[3] Joachim Biskup and Torsten Polle. Decomposition of Database Classes under Path Functional Dependencies and Onto Constraints. In *Foundations of Information and Knowledge Systems*, pages 31–49, 2000.

[4] Joachim Biskup and Torsten Polle. Adding inclusion dependencies to an object-oriented data model with uniqueness constraints. *Acta Informatica*, 39:391–449, 2003.

[5] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 155–160, 2001.

[6] David DeHaan, David Toman, and Grant E. Weddell. Rewriting Aggregate Queries using Description Logics. In *Description Logics 2003*, pages 103–112. CEUR-WS vol.81, 2003.

[7] Martin Fürer. Alternation and the Ackermann Case of the Decision Problem. *L'Enseignement Math.*, 27:137–162, 1981.

[8] Minoru Ito and Grant Weddell. Implication Problems for Functional Constraints on Databases Supporting Complex Objects. *Journal of Computer and System Sciences*, 49(3):726–768, 1994.

[9] Vitaliy L. Khizder, David Toman, and Grant E. Weddell. Reasoning about Duplicate Elimination with Description Logic. In *Rules and Objects in Databases, DOOD 2000 (part of Computational Logic 2000)*, pages 1017–1032, 2000.

[10] Vitaliy L. Khizder, David Toman, and Grant E. Weddell. On Decidability and Complexity of Description Logics with Uniqueness Constraints. In *International Conference on Database Theory ICDT'01*, pages 54–67, 2001.

[11] Huizhu Liu, David Toman, and Grant E. Weddell. Fine Grained Information Integration with Description Logic. In *Description Logics 2002*, pages 1–12. CEUR-WS vol.53, 2002.

[12] Carsten Lutz, Carlos Areces, Ian Horrocks, and Ulrike Sattler. Keys, Nominals, and Concrete Domains. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 349–354, 2003.

[13] Lubomir Stanchev and Grant E. Weddell. Index Selection for Embedded Control Applications using Description Logics. In *Description Logics 2003*, pages 9–18. CEUR-WS vol.81, 2003.

[14] David Toman and Grant E. Weddell. On Attributes, Roles, and Dependencies in Description Logics and the Ackermann Case of the Decision Problem. In *Description Logics 2001*, pages 76–85. CEUR-WS vol.49, 2001.

[15] David Toman and Grant E. Weddell. On Reasoning about Structural Equality in XML: A Description Logic Approach. *Theoretical Computer Science*, 2004. ICDT 2003 special issue, in press.

[16] P. van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathe-matics*, pages 331–363. Marcel Dekker Inc., 1997.

[17] Grant Weddell. A Theory of Functional Dependencies for Object Oriented Data Models. In *International Conference on Deductive and Object-Oriented Databases*, pages 165–184, 1989.

[18] Grant E. Weddell. Reasoning about Functional Dependencies Generalized for Semantic Data Models. *TODS*, 17(1):32–64, 1992.

# SONIC — System Description*

Anni-Yasmin Turhan and Christian Kissig
Institute for Theoretical Computer Science,
TU Dresden, Germany
*lastname*@tcs.inf.tu-dresden.de

**Abstract**

SONIC[1] is the first prototype implementation of non-standard inferences for Description Logics that can be used via a graphical user interface. In addition to that our implementation extends an earlier implementation of the least common subsumer and of the approximation inference service to more expressive Description Logics, more precisely to Description Logics with number restrictions. SONIC offers these reasoning services via an extension of the graphical ontology editor OILED [4].

## 1 Introduction and Motivation

Inference problems for Descriptions Logics (DLs) are divided into so-called standard and non-standard ones. Well-known standard inference problems are satisfiability and subsumption of concept descriptions. For a great range of DLs, sound and complete decision procedures for these problems could be devised and some of them are put into practice for very expressive DLs in state of the art DL reasoners as FACT [15] and RACER [13].

Prominent non-standard inferences are the least common subsumer (lcs), and approximation. Non-standard inferences resulted from the experience with real-world DL ontologies, where standard inference algorithms sometimes did not suffice for building and maintaining purposes. For example, the problem of how to structure the application domain by means of concept definitions may not be clear at the beginning of the modeling task. Moreover, the expressive power of the DL under consideration can make it difficult to come up with a faithful formal definition of the concept originally intended. This kind of difficulties can be alleviated by the use of non-standard inferences in the *bottom-up* construction of DL knowledge bases, as described in [1, 8]. Here instead of directly defining a new concept, the knowledge engineer introduces several typical examples as objects, which are then automatically generalized into a concept description by the DL system. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept. The task of computing

* This work has been supported by the Deutsche Forschungsgemeinschaft, DFG Project BA 1122/4-3.

[1] SONIC stands for "Simple OILED Non-standard Inference Component".

such a concept description can be split into two subtasks: computing the most specific concepts of the given objects, and then computing the least common subsumer of these concepts.

The lcs was first mentioned as an inference problem for DLs in [12]. Given two concept descriptions $A$ and $B$ in a description logic $\mathcal{L}$, the *lcs* of $A$ and $B$ is defined as the least (w.r.t. subsumption) concept description in $\mathcal{L}$ subsuming $A$ and $B$. The idea behind the lcs inference is to extract the commonalities of the input concepts. It has been argued in [1, 8] that the lcs facilitates the "bottom-up"-approach to the modeling task: a domain expert can select a number of intuitively related concept descriptions already existing in an ontology and use the lcs operation to automatically construct a new concept description representing the closest generalization of these concepts. For a variety of DLs there have been algorithms devised for computing the lcs, see [1, 16, 10] for details.

Approximation was first mentioned as a new inference problem in [1]. The *approximation* of a concept description $C_1$ from a DL $\mathcal{L}_1$ is defined as the least concept description (w.r.t. subsumption) in a DL $\mathcal{L}_2$ that subsumes $C_1$. The idea underlying approximation is to translate a concept description into a typically less expressive DL. Approximation can be used to make non-standard inferences accessible to more expressive DLs so that at least an approximate solution can be computed. In case the DL $\mathcal{L}$ provides concept disjunction, the lcs of $C_1$ and $C_2$ is just the concept disjunction $(C_1 \sqcup C_2)$. Thus, a user inspecting this concept description does not learn anything about the commonalities between $C_1$ and $C_2$. Using approximation, however, one can make the commonalities explicit to some extent by first approximating $C_1$ and $C_2$ in a sublanguage of $\mathcal{L}$ which does not provide disjunction, and then compute the lcs of the obtained approximations in $\mathcal{L}$. Approximation has so far been investigated for a few DLs, see [7, 6].

Another application of approximation lies in user-friendly DL systems, such as the editor OilEd [4], that offer a simplified frame-based view on ontologies defined in an expressive background DL. Here approximation can be used to compute simple frame-based representations of otherwise very complicated concept descriptions. OilEd is a widely accepted ontology editor and it can be linked to both state of the art DL reasoners, Racer [13] and FaCT [15]. Hence this editor is a good starting point to provide users from practical applications with non-standard inference reasoning services. The prototype system Sonic is the first system that provides some of the non-standard inference reasoning services via a graphical user interface and thus makes them accessible to a wider user group. Sonic was first introduced in [17] and it can be downloaded from `http://lat.inf.tu-dresden.de/systems/sonic.html`.

In the next section we give an application example which underlines that—although the supported DLs are much less expressive compared to the DLs supported by the current DL reasoners—the inferences implemented in Sonic can be useful in practice. In Section3 we turn to the implementation of Sonic and describe how the inferences are realized and and how Sonic is coupled to OilEd and the underlying DL reasoner Racer. Then we give an impression how users can work with Sonic and in the end we sketch how the Sonic prototype system can be extended in future versions.

## 2 An Application Example

Let us briefly recall the DLs covered by SONIC. Starting with a set $N_C$ of *concept names* and a set $N_R$ of *role names concept descriptions* are inductively defined with the help of a set of *concept constructors*. The DL $\mathcal{ALE}$ offers the top- ($\top$) and bottom-concept ($\bot$), concept conjunction ($C \sqcap D$), existential restrictions ($\exists r.C$), value restrictions ($\forall r.C$), and primitive negation ($\neg P, P \in N_C$ ). The DL $\mathcal{ALC}$ extends $\mathcal{ALE}$ by concept disjunction ($C \sqcup D$) and full negation ($\neg C$). Extending each of these DLs by number restrictions, i.e., *at most restrictions* ($\leq n\ r$) and *at least restrictions* ($\geq n\ r$) one obtains $\mathcal{ALEN}$ and $\mathcal{ALCN}$, respectively.

The semantics of these concept descriptions is defined in the usual model-theoretic way in terms of an *interpretation* $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$. The domain $\Delta_\mathcal{I}$ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^\mathcal{I}$ maps each concept name $P \in N_C$ to a set $P^\mathcal{I} \subseteq \Delta_\mathcal{I}$ and each role name $r \in N_R$ to a binary relation $r^\mathcal{I} \subseteq \Delta_\mathcal{I} \times \Delta_\mathcal{I}$. The semantics are extended to complex concept descriptions in the usual way, see for example [1, 6].

A *TBox* is a finite set of concept definitions of the form $A \equiv C$, where $A$ is a concept name and $C$ is a concept description. SONIC can only process TBoxes that are *unfoldable*, i.e., they are acyclic and do not contain multiple definitions. Concept names occurring on the left-hand side of a definition are called *defined concepts*. All other concept names are called *primitive concepts*.

Let us illustrate by an with an application example the procedure of computing the lcs for $\mathcal{ALEN}$-concept descriptions and the approximation of $\mathcal{ALCN}$-concept descriptions by $\mathcal{ALEN}$-concept descriptions. Consider the TBox $\mathcal{T}$ with $\mathcal{ALCN}$-concept descriptions modeling Airbuses and their configurations. $\mathcal{T}$ contains the following concept definitions:

$$
\begin{aligned}
\mathsf{Cargo\text{-}Config} &\equiv \neg\mathsf{Passenger\text{-}Config}\\
\mathsf{Airbus\text{-}300} &\equiv \mathsf{Plane}\\
&\quad \sqcap \exists \mathsf{has\text{-}configuration}.\mathsf{Cargo\text{-}Config}\\
&\quad \sqcap \exists \mathsf{has\text{-}configuration}.(\mathsf{Passenger\text{-}Config} \sqcap (\leq 2\ \mathsf{has\text{-}classes}))\\
\mathsf{Airbus\text{-}340} &\equiv \mathsf{Plane}\\
&\quad \sqcap (\geq 2\ \mathsf{has\text{-}configuration})\\
&\quad \sqcap \forall\, \mathsf{has\text{-}configuration}.(\mathsf{Passenger\text{-}Config} \sqcap (\geq 261\ \mathsf{has\text{-}seats}))\\
&\quad \sqcap \big(\exists \mathsf{has\text{-}configuration}.((\leq 419\ \mathsf{has\text{-}seats}) \sqcap (\leq 2\ \mathsf{has\text{-}classes})) \sqcup\\
&\qquad \exists \mathsf{has\text{-}configuration}.((\leq 380\ \mathsf{has\text{-}seats}) \sqcap (\leq 3\ \mathsf{has\text{-}classes}))\big)
\end{aligned}
$$

If we want to find the commonalities between the concepts Airbus-300 and Airbus-340 by using the lcs in $\mathcal{ALEN}$, we first have to compute the approximation of Airbus-340 since its concept definition contains a disjunction. After that can we compute the lcs of Airbus-300 and $approx_{\mathcal{ALEN}}$(Airbus-340).

In order to compute the approximation of the concept definition of Airbus-340 we first make implicit information explicit. In our example this is done by propagating the value restriction onto the two existential restrictions which yields the new disjunction:

$\exists$has-configuration.
(Passenger-Config $\sqcap$ ($\geq$ 261 has-seats) $\sqcap$ ($\leq$ 419 has-seats) $\sqcap$ ($\leq$ 2 has-classes)) $\sqcup$
$\exists$has-configuration.
(Passenger-Config $\sqcap$ ($\geq$ 261 has-seats) $\sqcap$ ($\leq$ 380 has-seats) $\sqcap$ ($\leq$ 3 has-classes)).

After the propagation of the value restriction, we can obtain the approximation of our example concept description by simply computing the lcs of these two disjuncts. They differ only w.r.t. the number occurring in the at-most restrictions. Consequently we have to pick the at-most restriction with the greater number from the two disjuncts and conjoin them with (Passenger-Config $\sqcap$ ($\geq$ 261 has-seats) to obtain their lcs. We get the following approximation of Airbus-340:

$approx_{\mathcal{ALEN}}$(Airbus-340) =
Plane $\sqcap$ ($\geq$ 2 has-configuration)
$\sqcap$ $\forall$ has-configuration.(Passenger-Config $\sqcap$ ($\geq$ 261 has-seats))
$\sqcap$ $\exists$has-configuration.
(Passenger-Config $\sqcap$ ($\geq$ 261 has-seats) $\sqcap$ ($\leq$ 419 has-seats) $\sqcap$ ($\leq$ 3 has-classes))

To compute the lcs of $approx_{\mathcal{ALEN}}$(Airbus-340) and the concept definition of Airbus-300, we need to unfold the concept Airbus-300 w.r.t. $\mathcal{T}$ by replacing Cargo-Config with its concept definition ¬Passenger-Config. It obvious now that the concept description implies two distinct configurations, since one of the existential restriction requires Passenger-Config and the other one ¬Passenger-Config. Thus the concept definition of Airbus-300 induces ($\geq$ 2 has-configuration) which occurs in $approx_{\mathcal{ALEN}}$(Airbus-340) directly. The primitive concept Plane appears in both concept descriptions and thus also in their lcs. Since the concept definition of Airbus-300 does not imply a value restriction the lcs does not contain any. Furthermore, we have to compute the lcs of each of the two existential restrictions from Airbus-300 and the one from $approx_{\mathcal{ALEN}}$(Airbus-340). We obtain for the overall lcs:

lcs($approx_{\mathcal{ALEN}}$(Airbus-340), Airbus-300) =
Plane $\sqcap$ ($\geq$ 2 has-configuration) $\sqcap$ $\exists$has-configuration.$\top$
$\sqcap$ $\exists$has-configuration.(Passenger-Config $\sqcap$ ($\leq$ 3 has-classes)).

The first existential restriction is redundant and therefore can be omitted—it would not be returned by our implementation. We have now extracted the commonalities of the Airbus-340 and the Airbus-300: they are both planes with at least two configurations where one configuration is a passenger configuration with up to 3 classes.

Although the DLs under consideration are not very expressible compared to the DLs handled by the state of the art DL reasoners this application example shows that an implementation of lcs and approximation for these DLs can be very useful to help users to extend their ontologies.

## 3 The SONIC Implementation

The SONIC system implements the algorithms for computing the lcs for $\mathcal{ALEN}$-concept descriptions and the approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions. Fur-

thermore, SONIC implements a graphical user interface to offer these non-standard inferences and an interface to a DL reasoner needed for subsumption queries.

### 3.1 Implementing the Inferences

We briefly sketch the main idea of the inference algorithms here. The algorithm for computing the lcs in $\mathcal{ALEN}$ was devised and proven correct in [16], thus our implementation is well-founded. The algorithm for computing the lcs of $\mathcal{ALEN}$-concept descriptions consists of three main steps:

1. *Unfold* the input concept descriptions by recursively replacing defined concepts by their definitions from the TBox.

2. *Normalize* the unfolded concept descriptions to make implicit information (e.g. inconsistencies, induced existential restrictions, induced value restrictions or induced number restrictions) explicit.

3. Represent the normalized concepts as concept trees, build the cross-product of the trees and read out a concept description from it.

For the DL $\mathcal{ALEN}$ the normalization and the structural comparison are much more involved than in $\mathcal{ALE}$. Firstly, the number restrictions for roles, more precisely the at most restrictions, necessitates merging of role-successors mentioned in the existential restrictions. To obtain all valid mergings is a combinatorial problem. Second, the commonalities of all mergings for existential restrictions of a concept description have to be determined by computing their lcs. These in turn are then used to compute the cross-product. In our implementation we use lists as data structures to represent the concept descriptions and implement the algorithms without advanced optimizations in order to keep this first implementation of the lcs for $\mathcal{ALEN}$-concept descriptions simple and easy to test.

The lcs algorithm for $\mathcal{ALEN}$ can return concept descriptions double exponential in the size of the input concepts in the worst case. Nevertheless, so far the lcs in $\mathcal{ALEN}$ realized in SONIC is a plain implementation of this algorithm. Surprisingly, a first evaluation of our implementation shows that for concepts of an application TBox with only integers from 0 to 5 used in number restrictions the run-times remained under a second (with Allegro Common Lisp on a Pentium IV System, 2 GHz). Our implementation of the lcs for $\mathcal{ALE}$-concept descriptions as described in [3] uses lazy unfolding. Due to this technique shorter and thus more comprehensible concept descriptions can be obtained more quickly, see [3]. To implement lazy unfolding for the lcs for $\mathcal{ALEN}$-concept descriptions in SONIC is yet future work.

The algorithm for computing the approximation of $\mathcal{ALCN}$-concept descriptions by $\mathcal{ALEN}$-concept descriptions was introduced and proven correct in [6]. The idea underlying this algorithm is similar to the lcs algorithm. For approximation the normalization process additionally has to build a disjunctive normal form on each role-level by "pushing" the disjunctions outward. With concept descriptions in this normal form the commonalities of the disjuncts are computed by applying the lcs on each role-level.

The approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions is also implemented in Lisp and uses the above mentioned implementation of the lcs for $\mathcal{ALEN}$-concept descriptions as a subfunction. A first implementation of the approximation of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions is described in [7]. This implementation is now extended to number restrictions and provided by SONIC.

In the worst case the approximation in both pairs of DLs can yield concept descriptions that are double exponential in the size of the input concepts descriptions. Nevertheless, this is not a tight bound. A first evaluation of approximating randomly generated concept descriptions shows that, unfortunately, both implementations run out of memory already for concepts that contain several disjunctions with about 6 disjuncts. It is unknown whether this kind of concept descriptions appears in application TBoxes from practical applications. Nevertheless, effective optimization techniques are needed for computing approximation, before this service can be applied to large ontologies. Similar to the lcs one can apply lazy unfolding to avoid "unnecessary" unfolding and thereby obtain smaller concept descriptions even faster. Besides lazy unfolding there is also the approach of so called *nice concepts* described in [9] known as an optimization technique for approximation. Currently these techniques are implemented and evaluated for approximation of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions in a student's project in our group.

The implementation of the algorithms for both inferences are realized in a straightforward way without sophisticated data structures or advanced optimizations as, for example the caching of results. This facilitated the testing and debugging of the SONIC prototype.

### 3.2 Linking the System Components

In order to provide the inferences lcs and approximation to users of the ontology editor OILED, we need not only to connect to the editor OILED, but also to a DL reasoner since both inferences, lcs and approximation, use subsumption tests heavily during their computation. The connection from SONIC to the editor OILED, is realized as a plug-in. Like OILED itself, this plug-in is implemented in Java. SONIC's plug-in is implemented for OILED version 3.5.3 (or higher) and realizes mainly the graphical user interface of SONIC. A screen-shot of the lcs tab in SONIC is shown in Figure 3.2. SONIC's Java plug-in connects via a Telnet connection to the Lisp implementation of the non-standard inferences to pass concept descriptions or messages between the components.

The OILED user can classify an ontology in the OILED editor, by either connecting OILED to the FACT reasoner via a CORBA interface or to any DL reasoner supporting the DIG ("Description Logic Implementation Group") protocol. The DIG protocol is an XML-based interface for DL systems with a tell/ask syntax, see [5]. DL developers of most DL systems have committed to implement this standard in their system making it a standard for DL related software.

SONIC must have access to the same instance of the reasoner that OILED is connected to in order to have access to the information from the ontology, more precisely, to make use of stored concept definitions and of cached subsumption relations obtained

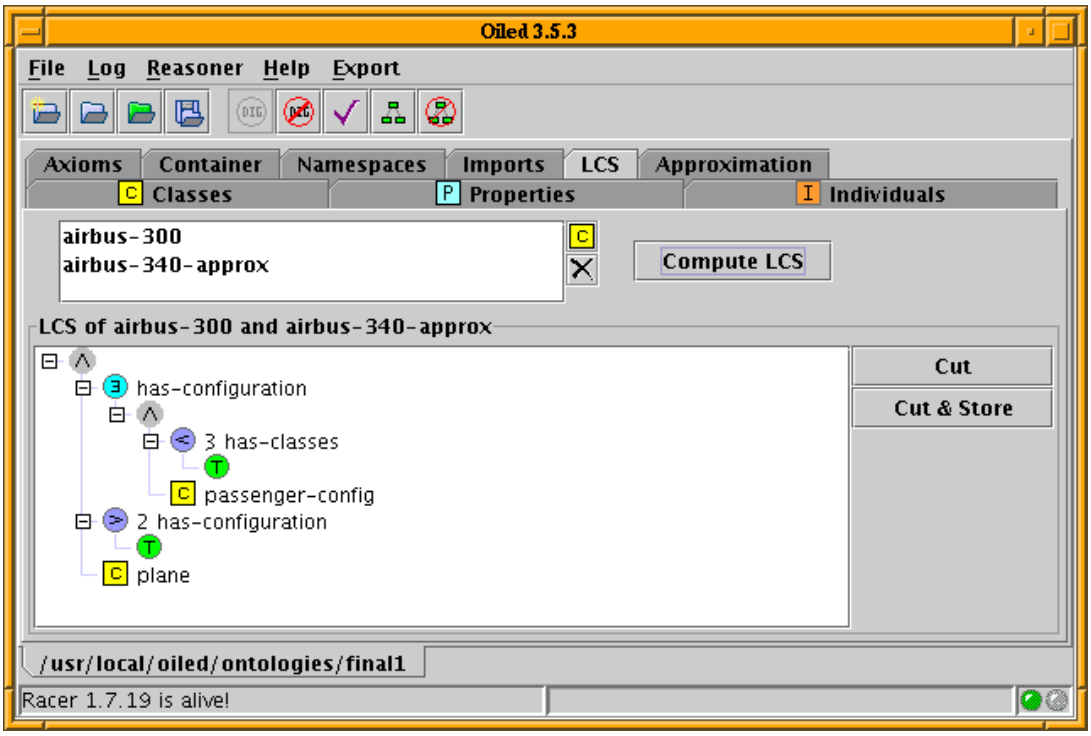



Figure 1: SONIC's lcs Tab in OILED.

during classification by the DL reasoner. If OILED and the DL reasoner do not have consistent versions of the ontology, the computed results for lcs and approximation might simply be incorrect due to this inconsistency.

SONIC needs the functionality of retrieving the concept definition of a concept defined in the TBox in order to perform unfolding. Since such a function is not included in the DIG protocol, we cannot use the DIG interface to connect to the DL reasoner. Since the CORBA interface to FACT is slow, we use RACER as underlying DL reasoner. SONIC connects to RACER version 1.7.7 (or higher) via the TCP socket interface described in [14]. Note, that in this setting the RACER reasoner need not run locally, but may even be accessed via the web by OILED and SONIC.

### 3.3 SONIC at Work

After the user has started the OILED editor with SONIC, the lcs and approximation inference are available on two extra tabs in OILED— as shown in Figure 3.2. After the OILED user has defined some concepts in the OILED ontology, has connected to the DIG reasoner RACER and classified the ontology, she can use, for example, the lcs reasoning service. In order to do so she can select some of the concept names from the ontology on the lcs tab. When the button 'compute lcs' is clicked, the selected concept names are transmitted to SONIC's Lisp component and the lcs is computed based on

the current concept definitions stored in RACER.[2] The concept description obtained from the lcs implementation is send to the plug-in via Telnet and displayed on the lcs tab. The approximation inference is offered on a similar SONIC tab in OILED.

Since the concept descriptions returned by the lcs and the approximation inference can be very large, it is not feasible to display them in a plain fashion. SONIC displays the returned concept descriptions in a tree representation, where uninteresting subconcepts can be folded away by the user and inspected later. In Figure 3.2 we see how the concept description for the lcs obtained from the application example in Section 2 is displayed on SONIC's tab in OILED. Based on this representation SONIC also provides limited functionality on both of its tabs to edit concept descriptions. OILED users can 'cut' subdescriptions from the displayed concept description and thereby reduce the displayed concept description to interesting aspects. OILED users can also 'cut and store' (a part of) the obtained concept description under a new concept name in their ontology.

## 4 Conclusions and Future Work

The SONIC prototype is a graphical tool for supporting main steps of the bottom-up approach for augmenting ontologies. These steps are realized by implementations of the least common subsumer in $\mathcal{ALEN}$ and the approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions. These reasoning services can be used from within the OILED ontology editor. Since SONIC is the first system that implements a graphical user interface for non-standard inferences, it is now possible to evaluate how useful these inference services are to users from practical applications.

Currently there is a big language gap between the DLs implemented in the state of the art DL reasoners and the DLs for which non-standard inferences are investigated or even implemented. To overcome this language gap to some extend we are currently studying a new approach to compute approximate solutions for the lcs and thus obtain a "good" common subsumer (instead of a least one) for input concept descriptions referring to concepts defined in a more expressive DL, see [2].

Developing SONIC is ongoing work. Our next step is to optimize the current implementation of approximation—on the one hand to speed-up the computation and on the other hand to obtain smaller concepts. This can be achieved by using lazy unfolding as our lcs implementation for $\mathcal{ALE}$ has shown, see [3]. Another step is to implement minimal rewriting w.r.t. TBoxes to obtain more concise and thus better comprehensible result concept descriptions from both reasoning services. In the longer run we want to comprise the implementations of the difference operator (see [7]) and of matching for $\mathcal{ALE}$ (see [11]) in SONIC and provide these inference services to users from practical applications. The SONIC system can be down loaded for research purposes from `http://lat.inf.tu-dresden.de/systems/sonic.html`.

Finally we would like to thank Ralf Möller, Volker Haarslev and Sean Bechhofer for their help on how to implement SONIC's linking to RACER and OILED.

[2] This is why the TBox should be classified first.

## References


[1] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, editor, *Proceedings of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann, Los Altos.

[2] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. In the Proceedings of 2004 International Workshop on Description Logics (DL 2004). To appear.

[3] Franz Baader and Anni-Yasmin Turhan. On the problem of computing small representations of least common subsumers. In *Proceedings of the 25th German Annual Conf. on Artificial Intelligence (KI'02)*, LNAI. Springer–Verlag, 2002.

[4] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proceedings of the 24th German Annual Conf. on Artificial Intelligence (KI'01)*, volume 2174 of *LNAI*, pages 396–408, Vienna, Sep 2001. Springer-Verlag.

[5] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG description logic interface. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, Rome, Italy, 2003.

[6] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In *Proceedings of the 2002 Description Logic Workshop (DL 2002)*, number 53 in CEUR-WS. RWTH Aachen, April 2002.

[7] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuinness, and M.-A. Williams, eds., *Proceedings of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-02)*, 2002. Morgan Kaufmann Publishers.

[8] Sebastian Brandt and Anni-Yasmin Turhan. Using non-standard inferences in description logics — what does it buy me? In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (KIDLWS'01)*, number 44 in CEUR-WS, Vienna, Austria, September 2001. RWTH Aachen.

[9] Sebastian Brandt and Anni-Yasmin Turhan. An approach for optimized approximation. In *Proceedings of the KI-2002 Workshop on Applications of Description Logics (KIDLWS'02)*, number 63 in CEUR-WS, Aachen, Germany, September 2002. RWTH Aachen.

[10] Sebastian Brandt, Anni-Yasmin Turhan, and Ralf Küsters. Extensions of non-standard inferences for description logics with transitive roles. In *Proceedings of the 10th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'03)*, LNCS. Springer, 2003.

[11] Sebastian Brandt. Implementing matching in $\mathcal{ALE}$—first results. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, CEUR-WS, 2003.

[12] Wiliam Cohen, Alex Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In W. Swartout, editor, *Proceedings of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, pages 754–760, San Jose, CA, 1992. AAAI Press/The MIT Press.

[13] Volker Haarslev and Ralf Möller. RACER system description. In *Proceedings of the International Joint Conference on Automated Reasoning IJCAR'01*, LNAI. Springer Verlag, 2001.

[14] Volker Haarslev and Ralf Möller. *RACER User's Guide and Manual*, Sept. 2003. available from: `http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-7.pdf`.

[15] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In A.G. Cohn, L.K. Schubert, and S.C.Shapiro, editors, *Proceedings of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 636–647, 1998.

[16] Ralf Küsters and Ralf Molitor. Computing Least Common Subsumers in $\mathcal{ALEN}$. In B. Nebel, editor, *Proceedings of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI-01)*, pages 219–224. Morgan Kaufman, 2001.

[17] Anni-Yasmin Turhan and Christian Kissig. SONIC—Non-standard inferences go OILED. In *Proceedings of the International Joint Conference on Automated Reasoning IJCAR'04*, LNAI. Springer Verlag, 2004. To appear.

# Understanding Ontologies in Scholarly Disciplines

Brian R Gaines
Knowledge Science Institute
University of Calgary, Alberta, Canada
gaines@ucalgary.ca

**Abstract**

Description logics are valuable for modeling the conceptual structures of scientific and engineering research because the underlying ontologies generally have a taxonomic core. Such structures have natural representations through semantic networks that mirror the underlying description logic graph-theoretic structures and are more comprehensible than logical notations to those developing and studying the models. This article reports experience in the development of visual language tools for description logics with the objective of making research issues, past and present, more understandable.

## 1 Introduction

Scholarship may be conceptualized as the rational reconstruction of intuitive notions within the conventions of a discipline. When scholarly disciplines examine their foundations the outcome is generally a taxonomy based on logical definitions intended to capture the concepts of the primary researchers and to clarify the differences underlying disagreements. The development and analysis of such taxonomies can be helpful to active research communities attempting to clarify their activities, and it is also significant in retrospect to historians reconstructing the conceptual structures of those recognized as major contributors to the growth of human knowledge. Description logics managed through visual languages isomorphic to the underlying graph-theoretic structures, and visually transformable through well-defined deductive processes, offer an attractive technology to support both historic studies and active research communities.

The work reported in this paper is a continuation of that on the use of knowledge acquisition and representation tools to model the knowledge structures of scholarly communities [1]. These studies involved the use of the visual language [2] that allowed knowledge structures to be expressed as semantic networks with well-defined semantics that were automatically translated into data structures in KRS [3], an implementation of a CLASSIC-like [4] description logic. Inferences in KRS were graphed automatically as additions to the semantic network so that users could visualize both the inputs and outputs without translation into logical formulae [5].

In recent years there have been major advances in description logic research that make it realistic to use richer representations incorporating negation, disjunction and some aspects of recursion [6]. This enables one to overcome of the artificiality of the knowledge structures noted above that attempted to avoid such constructions, resulting in unnatural representations, of lesser use as models meaningful to the relevant community.

This paper reports on recent developments that: extend the visual language to support richer description logics with disjunction, negation and existential quantification; exemplifies the process of transforming semantic networks in coming to understand them; discusses factoring deduction into its intensional and extensional operations to support paraconsistent reasoning [7]; and raises a number of issues for future research.

# 2 Understanding an Ontology

I have used the term "understanding an ontology" in order to capture the notion that users should be able to see the effects of variations in the ontology, some of which do not change its meaning, others of which change it significantly in ways that can be readily understood, and others of which are logical consequences that may require some degree of explanation if they are to be understood.

A visual language representation of ontologies is useful to support those without great fluency in symbolic logic in its textual representation. Shin [8] has demonstrated that diagrammatic reasoning can provide a rigorous foundation for logical inference, and psychological experiments show that non-technical users of a knowledge-based system find inference in a visual language easier to understand [9].

## 2.1 Designing a visual language for description logics

The design criteria for the visual language have been:

1. The visual language should provide an alternative syntax to linear textual languages but have standard logical semantics and be intertranslatable with textual languages.
2. The visual language should be simple to explain to users, both those with deep understanding of symbolic logic and those with little understanding.
3. The visual language should correspond to the natural graph-theoretic representation of description logics that is commonly used in describing operations on them and in implementing computational inference [10].
4. As many as possible of the syntactic and inferential transformations of expressions in the visual language should be formally specifiable as graph-theoretic operations.
5. The visual language should be usable both for the input of logical definitions and assertions, and for the output of logical inferences.
6. The visual language should support modularity in the specification of ontologies such that definitions/assertions may be specified in one document and used in others [11].
7. Subject to these requirements, the visual language should be similar to existing languages for semantic networks.

## 2.2 Overview of the KNet visual language

KNet, the visual language used in this paper is implemented in a generic visual language shell, RepNet [12], that supports a user-specified syntax for node types and connecting lines and a user scriptable interface for translation to and from semantic networks in the visual language enabling integration with web services such as KRS and RACER [13]. It is simple to change the language to conform to existing practices, user preferences, or changing notions of what is required. The examples given follow the conventions described in [2] and are similar to those of other graphical interfaces for description logics such as RICE [14].

**Concepts** are represented by the concept name in an oval. Concepts are *defined* through the property they *encode* [15] in the graph derived by tracing outgoing arrows from the concept, terminating at concept nodes or terminal nodes. Concepts are *used* through incoming arrows, and may be both defined and used in the same graph.

**Base**, or primitive, concepts are indicated by short horizontal markers in the concept oval that indicate that there is some unspecified outgoing graph unique to the concept. This is a sufficient *explication* (in Carnap/Quine terms [16]) for the logical properties of a primitive concept. From a logical perspective, is does not matter in reasoning with the concept how the unspecified graph is represented provided it is unique to the concept. However, in modeling scientific reasoning, one has to take into account that each school of thought may have

adopted a differing, more specific representation of a particular primitive concept, making a distinction without a difference that can be a source of confusion in the literature.

**Roles**, or relations, are represented by the role name without any surrounding shape.

**Individuals**, or singletons, are represented by the individual name in a rectangle. An individual *exhibits* [15] a property derived from its outgoing arrows as for concepts, which may be conceptualized as the concept encoding its current *state*.

A collective individual or **set** is represented by an extensional constraint, and possibly an identifying name, in a rectangle with inset vertical lines at each end. The constraint is specified through upper and lower cardinality and inclusion bounds on the collection as detailed in [3] where it was shown that such bounds may be conceptualized as defining generalized sets or mereological collections having well defined unions, intersections and complements, and forming a subsumption lattice under inclusion ordering (I have not yet found an elegant graphical representation of the bounds, and hence have left these defined in textual form within the node). Sets are important in representing role fillers, co-reference and inclusion constraints, and, when defined by comprehension, material implications or rules. The notation for an individual may be regarded as a shorthand for a set with cardinality 1 (consistent with the Quine/Scott [17/18] extensional simplification of set theory that a={a}). Thus there are basically only three types of nodes: concepts, roles and mereological sets. The node type "∃" is provided as a shorthand for the cardinality constraint "≥1".

**Arrows** between nodes derive their semantics from the types of the nodes they connect.

An arrow from concept A to concept B means that concept A is defined to be **subsumed** by concept B. The equivalent graph-theoretic interpretation is that the arrow may be replaced by copying the graph of outgoing arrows from concept B to concept A (including the unspecified graph of a primitive concept).

An arrow from individual A to concept B means that A is asserted to be an **instance** of B and again may be given a graph-theoretic interpretation as a copy operation.

An arrow from an individual A or a set A to a set B means that A is contained in B. This can be used to specify co-reference and inclusion constraints. An arrow from a concept A to a set B means that any individual comprehended by A is contained in B. This has the corollary that a **rule**, or material implication, may be represented as a set with a incoming arrow from a premise concept and an outgoing arrow to a conclusion concept.

Multiple arrows from a node are taken as specifying a **conjunction** of properties. This convention necessitates the introduction of a special node, "∨", specifying a **disjunction**, with the convention that outgoing arrows from this specify a disjunction of properties. The graph-theoretic interpretation is one of multiple alternative graphs each having one of the branches of the disjunction, and disjunction nodes can always be eliminated by such expansion resulting in multiple, alternative definitions of a disjunctive concept.

The conjunctive node type "∧" is also available to use after a "∨" to disambiguate multi-branch outgoing graphs that are to be treated as a single term in the disjunction.

**Negation** is represented through an arrow with a cross bar having the graph-theoretic interpretation that the graph at the end of the arrow must *not* occur. This gives rise to the standard semantics for negation, including De Morgan's laws linking conjunction, disjunction and negation. A negation arrow from a concept to a set may be used to represent a rule with exceptions [19].

An **existential** constraint is specified through a set with an arrow to a concept applying to the individuals included in it.

If a graph contains a conjunction/disjunction of two identical graphs then one of the conjuncts/disjuncts may deleted.

### 2.3 Models, satisfaction and subsumption

A concept definition is **coherent**, or consistent, if all the set bounds specified in it are consistent and there is no conjunction in it of an arrow and a negation arrow pointing to the same graph.

A **model** satisfying an ontology defined in the visual language is a collection of individuals satisfying all the existential constraints such that their resulting states are coherent.

One ontology is **extensionally subsumed** by another if any model satisfying it also satisfies the other. This definition gives rise to the standard denotational, extensional, model-theoretic semantics for description logics, and may be used to show that the graph-theoretic operations of the visual language conform with the standard extensional semantics of description logics.

We may also introduce the notion of intensional or structural subsumption as a sub-graph relation, that one ontology is **intensionally subsumed** by another if that other ontology is a sub-graph of it. It follows immediately that intensional subsumption implies extensional subsumption, but not necessarily *vice versa*.

However, the definition of intensional subsumption needs strengthening. First, a semantic network may be conceptualized as a *meta-graph* that specifies a set of equivalent graphs derivable from it by expansion, contraction and other logical operations. One ontology intensionally subsumes another if its graph at any stage of expansion or contraction is a sub-graph of the other at any stage of expansion or contraction. One could state this in terms of full expansions to a canonical form but for computational purposes the definition given is more useful, particularly since recursive definitions give rise to infinite graphs.

Second, the labels given to non-primitive concepts are arbitrary from a logical perspective, so that any remapping of labels that preserves non-identity may be used in computing structural subsumption. This corresponds to the notion that different terms are being used for the same concept, and is important in the analysis of scientific definitions since it often happens that different terminology has been used for essentially the same concept. Mapping primitives to one another is a deeper operation since it would imply that their tacit definitions are the same, and is also important to the process of finding explications of the primitives. A good example is the way in which the application of biological evolutionary theory to processes in other disciplines has led to the abstraction of the principles of variety generation and selective filtering underlying a general process of 'evolution.'

Third, the semantics of set constraints have not been specified in graphical form, but their subsumption lattice is well-defined so that one needs to extend the notion of sub-graph to be one in which a set matches another if it subsumes it.

## 3 Some Examples

In order to illustrate some of the issues, this section takes the following simple ontology from *The Description Logic Handbook* [6, p.52] and shows how it may be manipulated in KNet.

| | | | |
|---|---|---|---|
| Woman | ≡ | Person ⊓ Female | (1) |
| Man | ≡ | Person ⊓ ¬Woman | (2) |
| Mother | ≡ | Woman ⊓ ∃ has_Child.Person | (3) |
| Father | ≡ | Man ⊓ ∃ has_Child.Person | (4) |
| Parent | ≡ | Father ⊔ Mother | (5) |
| Grandmother | ≡ | Mother ⊓ ∃ has_Child.Parent | (6) |
| Mother_With_Many_Children | ≡ | Mother ⊓ ≥3 has_Child | (7) |
| Mother_Without_Daughter | ≡ | Mother ⊓ ∀ has_Child.¬Woman | (8) |
| Wife | ≡ | Woman ⊓ ∃ has_Husband.Man | (9) |

**Figure 1 Simple ontology of family relationships [6, p.52]**

Fig.2 shows the ontology of Fig.1 represented as a semantic network in KNet.

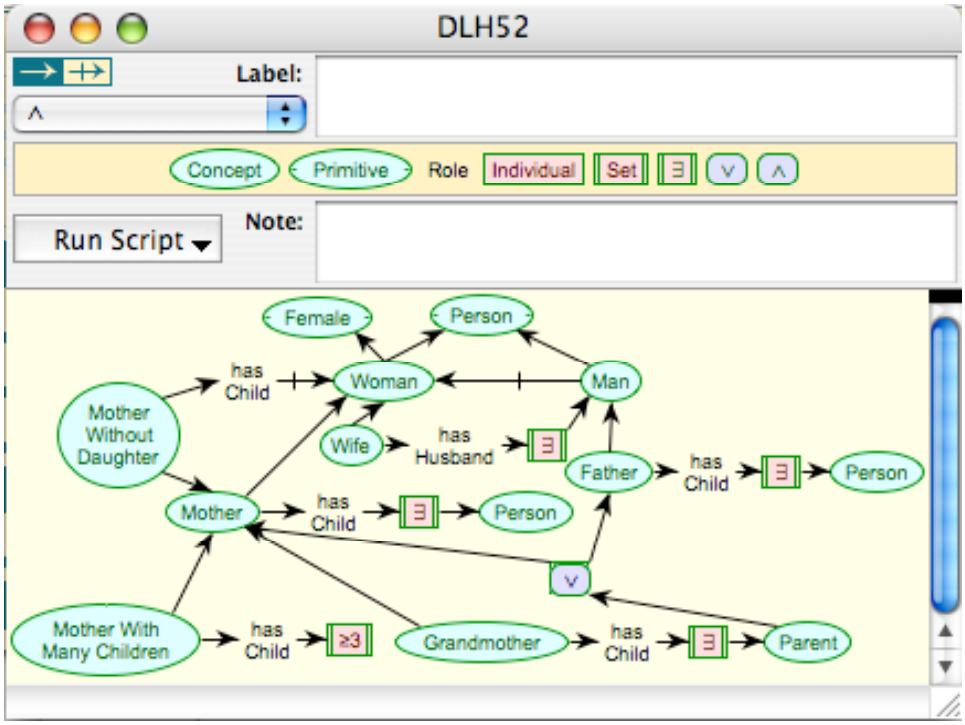



**Figure 2 Ontology of Figure 1 in KNet**

Figure 3 is equivalent to Figure 2, and derived from it by expanding all defined constructs, pushing negation to terminal nodes, and excising contradictory branches from disjunctions. These are all transformations that a representation system will probably make in transforming the definitions into an internal normal form, and they are also of help to the user in understanding the ontology.

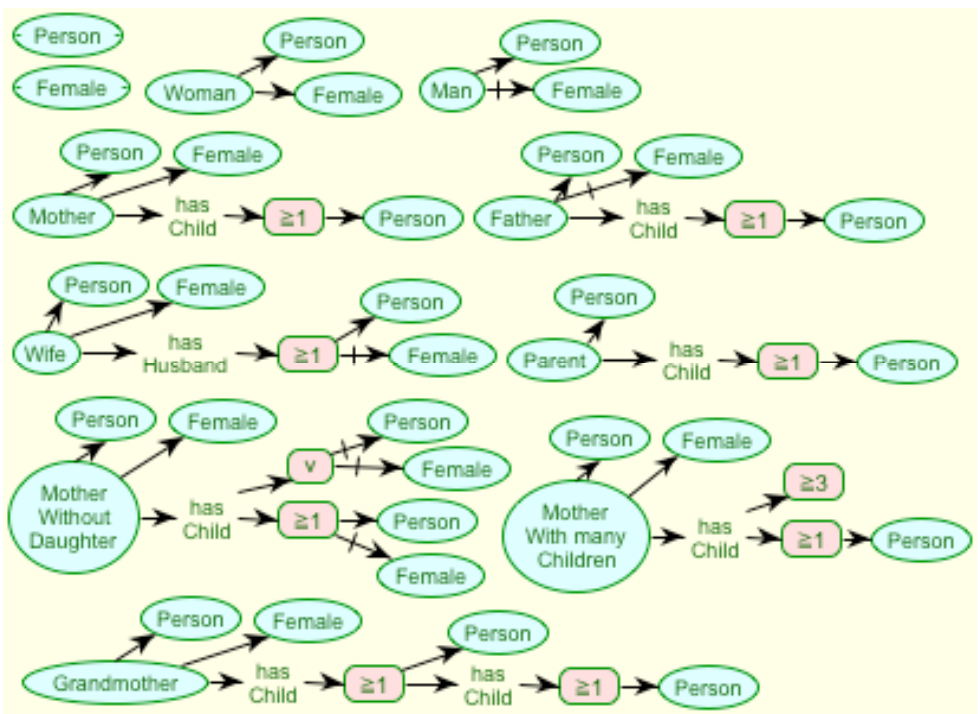



**Figure 3 Expanded ontology of Figure 2**

Some problems with the ontology defined in Figure 2 are apparent in Figure 3. "Mother Without Daughter" and "Mother With Many Children" are not defined as expected because they encompass situations in which a child is not a person. The problem may be viewed as arising from the definition of "Mother" that has "Person" after an existential quantifier, and it could be avoided by moving "Person" back to be a universal quantifier of the "has Child" role. However, this would have the consequence in recognizing a "Mother" that all her children would have to be checked to be people when it is intended that only the existence of one need be checked.

These problems are arising because the ontology of Figure 2 avoids the use of the natural recursive definition that the "has Child" role of a person must be filled by a person. However, this is an innocuous use of recursion since the concept "Person" is a primitive that can only be asserted of an individual, not recognized as applying to it, and hence the recursive definition acts only as a constraint that needs propagating through a graph up to its existing terminal nodes, not expanded indefinitely beyond them.

Fig. 4 shows an alternative ontology with the recursive definition, and Fig. 5 shows that it leads to the expected definitions after expansion.

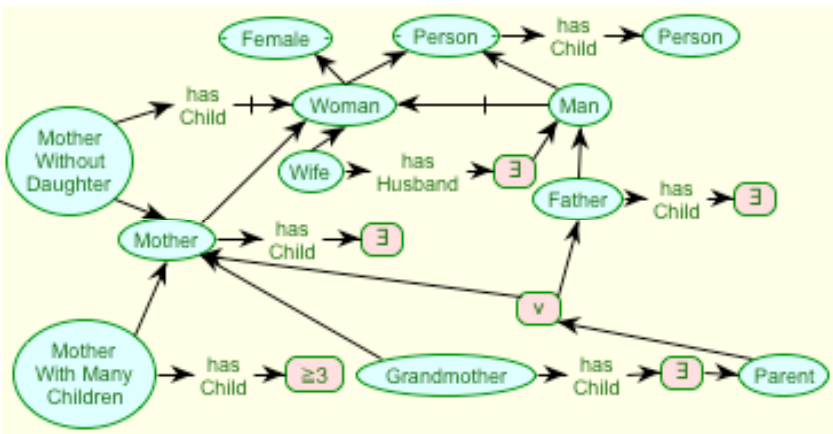



**Figure 4 Alternative ontology to Figure 1**

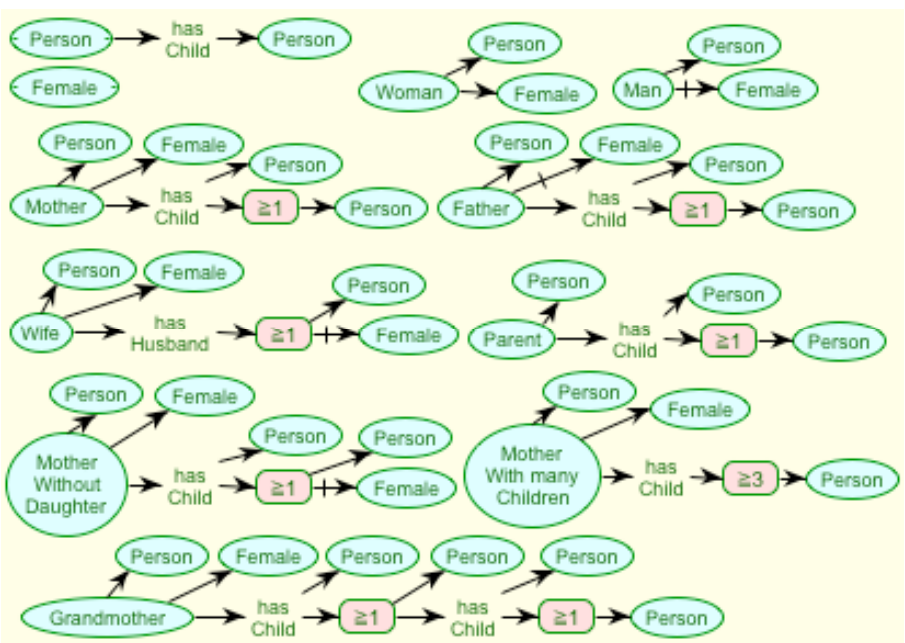



**Figure 5 Expanded ontology of Figure 4**

Fig. 5 looks somewhat cluttered with "Person" terminals, and the user might wish to limit the expansion by specifying that those implicit in the recursive definition are not shown, in effect that the "has Child" role of a person is implicitly filled by a person. Nine uses of "Person" can be dropped in Fig. 5 while preserving its equivalence to Fig. 4.

The point of this discussion to illustrate how various transformations of a ontology may affect the understanding of it, and need to be supported through decision logic inference and graphical interaction. Users need to be able to move back and forth between readily understood equivalent representations, much as does the inference engine. It is interesting to see how the defined subsumptions in Fig.4 are clearly visible to users as inferable subsumptions in Fig. 5 through subgraph relationships. Users also need to be able to compare the effects of changes that do affect meaning such as those between Fig.2 and Fig.4.

The situation becomes more complex as inferences are made that go beyond the restructuring discussed so far, for example, if it is the A-Box that is being graphed and extensional case-by-case reasoning has been applied or rules have fired. KRS graphs the results of such reasoning as additions as to the original graph after "infer" nodes, but makes no attempt to "explain" them. CLASSIC provides a form of explanation of terminological reasoning [20] and this together with more recent developments [21,22] suggest approaches which it would be interesting to implement as modules providing graphical output through semantic networks.

In its applications to presenting output from clustering algorithms, KNet provides an interactive interface whereby users can adjust what parts of a graph are shown by moving a slider to change a threshold. It would be interesting to take output from an inference engine in a *proof markup language* [23] and have a slider that moved through a linear representation of the proof steps while showing the resulting inferences being graphed in the semantic network.

Figure 6 provides a simple example of the distinctions made in Aristotle's mechanics that led to problems that medieval scientists attempted to resolve with little progress until their reconstruction by Galileo facilitated their explication by Newton [24]. The state of an object was seen to be either one of rest or one of motion, and several states of motion were distinguished. One that was well-grounded in experience but problematic in the development of a unified science of motion, was the distinction between heavenly and local bodies. The problems generated by this distinction without a difference were greatly exacerbated by making circular, constant motion part of the definitional essence of heavenly motion, thus requiring no explanation in terms of material conditionals or 'laws of motion.'

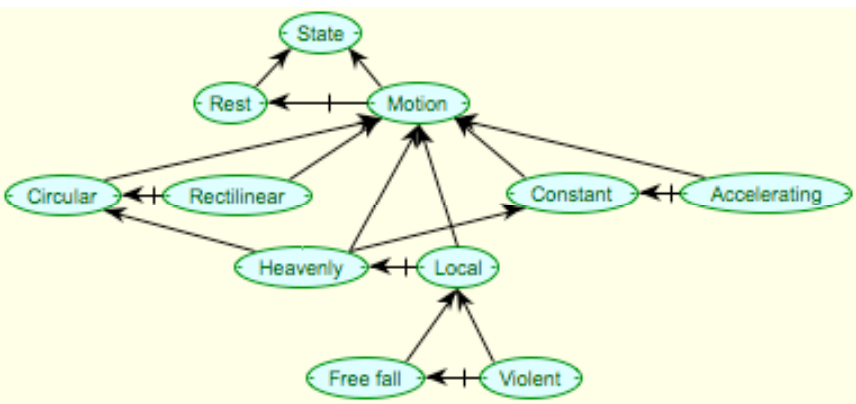



**Figure 6 Distinctions in medieval mechanics**

Medieval scientists accepted the heavenly—local distinction and the lack of need to explain the 'perfection' of heavenly motion, and focused on problems with the behavior of bodies in free-fall motion, that they accelerated, and ones in 'violent' motion, such as projectiles, that they continued in the direction in which they were projected even though they had lost contact with the projector. This led to explanations in terms of notions such as impressed 'impetus' [24].

Galileo dropped the distinction between heavenly and local bodies and between free fall and violent motion, but introduced new problems through the notion that the earth itself was moving and yet this had no apparent effect on objects in free fall. One can track the changing ontologies and laws from Aristotle through Buridan and Oresme to Copernicus and Galileo and hence to Descartes, Huygens, Hooke, Newton *et al*, as the gradual reduction of primitives in the ontologies of motion and their replacement by the material implications which we now know as Newton's 'laws of motion' [24].

# 4 The Transition from Logical Opposition to Numeric Scales

One important phenomenon in the development of scientific reasoning is the way in which qualitative distinctions become refined to be graded distinctions, eventually becoming numeric scales of observable measured with ever-increasing precision [25]. We can model this process in a description logic by introducing the natural symmetry of an opposition, that it is generally conceived as based on two opposing concepts of equal status rather than one and its negation as in Figs. 2, 4 and 6. The resultant structure turns out to have the properties of a multi-point scale.

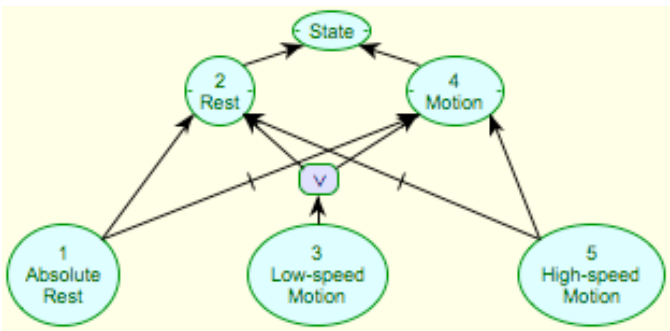



**Figure 7 From an opposition to a five-point scale**

Fig. 7 exemplifies this process. The opposition between the primitives, rest and motion, is modeled by the extremes of absolute rest and high-speed motion which inherit from one concept and the negation of the other. This leads to a natural five-point scale as three other concepts are interpolated between them, the two primitives and their disjunction. Seven and nine-point scales may be developed from this by adding another concept such as 'extreme value.' Once the logical possibility of grading the opposition has been realized it is natural to look for quantities to measure that correlate with the scale and provide further gradations.

# 5 Supporting Paraconsistent Reasoning

In the literature on modeling scientific reasoning it has been argued that inconsistencies are often present but do not cause "explosive" growth of conclusions through the *ex falso quodlibet* derivations of classical logic [7]. Hence it has been proposed that paraconsistent logics are needed to account for scientific reasoning [26]. However, uniform paraconsistency is not desirable since many major achievements in the scientific literature, such as Arrow's impossibility theorem, result from proofs of definitional inconsistency, and Rips' psychological studies show that people readily generate *reductio ad absurdum* arguments to solve logic problems [27].

Batens has proposed and developed *adaptive logics* that default to classical behavior in the absence of inconsistency, but behave paraconsistently in its presence [28]. Description logics are well-suited to be foundations for such logics if the reasoning is factored appropriately. The major example of *ex falso quodlibet* in description logics is that any incoherent definition is subsumed by any other. However, structural subsumption based on graph-matching does not lead to this conclusion. It has to be imposed separately. KRS [3] would happily report that a

Meinongian *green, round, square* entity, where *round* and *square* were declared disjoint primitives, was subsumed by *green*, *round* or *square* but not by *red*, provided that the additional inference step of noting that the definition was incoherent and mapping it to bottom was not taken.

Tableaux proofs by refutation obviously rely on such mapping but extensions to tableaux methods have been described which support inconsistency-adaptive logics [29] and it would be interesting to investigate how these might be incorporated in description logics.

A reasonable target architecture might be an inference engine with a user-interface through semantic networks and control over the proof methods such that one can see the impact of various methods in terms of the proofs generated and the inferences made. Normalization and structural subsumption might provide a model of the inference patterns that have led to incoherent definitions being accepted by scholarly communities for long periods of time, with inferences being made despite the contradictions. Stronger proof methods might provide a model of the anomaly detection that leads to a change in the conceptual framework marking a minor or major "scientific revolution."

# 6 Conclusions

Description logic technology, with visual language interfaces and control of proof techniques, provides very valuable tools for understanding the knowledge processes of current and past scholarly communities. Much of the current research on the support of the semantic web is directly applicable since the sub-disciplines of science are known to form a "semantic web" of inter-dependencies and provenances. It may be that some additional constructions will be needed, but the advances of recent years in description logic research make it reasonable to expect that it will be feasible to add them.

# References


[1] Brian R Gaines and Mildred L G Shaw. Using knowledge acquisition and representation tools to support scientific communities. In Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94), pages 707-714 1994.

[2] Brian R Gaines, An interactive visual language for term subsumption languages. In Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91), pages 817-823, 1991.

[3] Brian R. Gaines. A class library implementation of a principled open architecture knowledge representation server with plug-in data types. In Proc. of the 13th Int. Joint Conference on Artificial Intelligence (IJCAI'93), pages 504-509, 1993.

[4] A. Borgida, A., R. J. Brachman, D.L. McGuiness and L.A. Resnick. CLASSIC: a structural data model for objects. In Proc. of SIGMOD Conference on the Management of Data, pages 58-67, 1989.

[5] Brian R Gaines and Mildred L G Shaw. Embedding formal knowledge models in active documents. Communications of the ACM **42**(1): 57-63, 1999.

[6] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.

[7] Walter A. Carnielli, Marcello E. Coniglio and Itala M. Loffredo D'Ottaviano (Eds.). Paraconsistency: The Logical Way to the Inconsistent. New York, Marcel Dekker, 2002.

[8] Sun-Joo Shin. The logical status of diagrams. Cambridge University Press, 1994.

[9] John T. Nosek and Itzhak Roth. A comparison of formal knowledge representations as communication tools: predicate logic vs semantic network. International Journal of Man-

Machine Studies 33, 227-239, 1990.

[10] Alex Borgida and Peter F. Patel-Shneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. Journal of Artificial Intelligence Research 1, 277-308, 1994.

[11] Brian R Gaines. A situated classification solution of a resource allocation task represented in a visual language. Int. J. Human-Computer Studies 40(2): 243-271, 1994.

[12] Rep IV Manual. Centre for Person-Computer Studies, 2004, http://repgrid.com.

[13] Volker Haarslev and Ralf Möller. Description of the RACER System and its Applications. Proc. of the Int. Workshop on Description Logics (DL-2001), pages 132-141, 2001.

[14] Ralf Möller, R. Cornet and Volker Haarslev. Graphical Interfaces for Racer: Querying DAML+OIL and RDF Documents. Proc. of the Int. Workshop on Description Logics (DL-2003), 2003.

[15] Edward N. Zalta. Intensional Logic and the Metaphysics of Intentionality. Cambridge, MA: MIT Press, 1988.

[16] Willard Van Orman Quine. Word and Object. Cambridge, MA: MIT Press, 1960.

[17] Willard Van Orman Quine. Set theory and its logic. Cambridge, MA: Harvard University Press, 1963.

[18] Dana S. Scott. Quine's individuals. In Ernest Nagel,Patrick Suppes and Alfred Tarksi (Eds.) Logic, Methodology and the Philosophy of Science. Stanford, CA: Stanford University Press, pages 111-115, 1962.

[19] Brian R Gaines. Integrating rules in term subsumption knowledge representation servers. In Proc. of the 9th National Conference on Artificial Intelligence (AAAI'91), pages 458-463.

[20] Deborah L. McGuiness and Alex Borgida. Explaining subsumption in description logics. In Proc. 14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95), pages 816-821, 1995.

[21] S. Schlobach and R. Cornet. Explanation of Terminological Reasoning: A Preliminary Report. Proc. of the Int. Workshop on Description Logics (DL-2003), 2003.

[22] Deborah L. McGuinness and Paulo Pinheiro da Silva. Infrastructure for Web Explanations. In Proc. of 2nd Int. Semantic Web Conference (ISWC2003), LNCS 2870, Berlin: Springer, pages 113-129, 2003.

[23] Paulo Pinheiro da Silva, Deborah L. McGuinness and Richard Fikes. Combinable Proof Fragments for the Web. Tech. Rep. KSL-03-04, Knowledge Systems Laboratory, Stanford University, 2003.

[24] Eduard Jan Dijksterhuis. The Mechanization of the World Picture. Oxford: Clarendon Press, 1961.

[25] M. Norton Wise (Ed.). The Values of Precision. NJ: Princeton University Press, 1995.

[26] Joke Meheus (Ed.). Inconsistency in Science. Dordrecht: Kluwer, 2002.

[27] Lance J. Rips. The Psychology of Proof: Deductive Reasoning in Human Thinking. Cambridge, MA: MIT Press, 1994.

[28] Diderik Batens, Chris Mortensen, Graham Priest and Jean-Paul Van Bendegem. Frontiers of Paraconsistent Logic. Baldock, UK: Research Studies Press, 2000.

[29] Diderik Batens and Joke Meheus. A Tableau Method for Inconsistency-Adaptive Logics. In Proc. Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000), LNCS 1847, Berlin:Springer, pages 127-14, 2000.

# Description Logics for *e*-Service Composition

Daniela Berardi
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
*berardi*@dis.uniroma1.it

*e*-Services represent a new model in the utilization of the network, in which self-contained, modular applications can be described, published, located and dynamically invoked, in a programming language independent way.

Research on *e*-Services spans over many interesting issues, including description, discovery, composition. Our research focuses on automatic composition synthesis.[1] More specifically, we have devised techniques that, given *(i)* a client specification (*target e-Service*) expressed as a transition system and *(ii)* a set of available *e*-Services, also described as transition systems, synthesizes a composite *e*-Service that *(i)* uses only the available *e*-Services and *(ii)* interacts with the client "in accordance" with the input specification. First, we have considered the situation where the target *e*-Service is completely specified, and then we have extended the framework to deal with a target *e*-Service which is underspecified and presents non-determinism, in the form of don't care condition on the next transitions.

Our techniques are based on reducing the problem of checking the existence of a composition into concept satisfiability in a knowledge base expressed in a Description Logic (DL) –or equivalently satisfiability of a formula in a theory expressed in a variant of Propositional Dynamic Logic. With this reduction, reasoning tools for DLs can be directly used for composition synthesis, in particular by extracting a composite *e*-Service from a model of the DL knowledge base. We have developed (and currently continue to extend) an open source prototype system[2], that realizes our techniques, which is, at the best of our knowledge, the first effective tool for automatic composition synthesis of *e*-Services that export their behavior.

[1] Relevant publications can be found at: `http://www.dis.uniroma1.it/~berardi/publications`.

[2] cfr. the PARIDE (Process-based frAmewoRk for composItion and orchestration of Dinamyc *E*-Services) Open Source Project: `http://sourceforge.net/projects/paride/`

# Description Logic and Order-sorted Logic

Ken Kaneiwa
National Institute of Informatics, Japan
kaneiwa@nii.ac.jp

I have been studying the following work on description logic, which is closely related to my main work on extensions of order-sorted logic and logic programming [4, 3, 2].

Title: Negation in Description Logics: Contraries, Contradictories and Subcontraries (under submission)

Abstract: We propose an alternative description logic $\mathcal{ALC}_{\sim}$ with classical negation and strong negation. In particular, we adhere to the notions of contraries, contradictories and subcontraries [1], generated from possible statement types using predicate denial and predicate term negation. To capture these notions, our formalization provides an improved semantics that suitably interprets various combinations of classical negation and strong negation (but not Heyting negation and strong negation). We develop a tableau-based satisfiability algorithm for $\mathcal{ALC}_{\sim}$, and show the correctness: soundness, completeness and termination and the complexity.

## References

[1] L. R. Horn. *A Natural History of Negation.* University of Chicago Press, 1989.

[2] K. Kaneiwa. Order-sorted logic programming with predicate hierarchy. *Artificial Intelligence*, 2004 (accepted).

[3] K. Kaneiwa and R. Mizoguchi. Ontological knowledge base reasoning with sort-hierarchy and rigidity. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, 2004.

[4] K. Kaneiwa and S. Tojo. An order-sorted resolution with implicitly negative sorts. In *Proceedings of the 2001 Int. Conf. on Logic Programming*, pages 300–314. Springer-Verlag, 2001. LNCS 2237.

# Explaining Description Logic Reasoning

Francis Kwong
University of Manchester, United Kingdom
francis.kwong@cs.man.ac.uk

The increasing reliance on automated reasoning needs not just the availability of fast reasoners, but also mechanisms to explain their, often surprising, results to human users, so that they can understand the consequences in their application domain. As one of the significant applications of machine reasoning is the Semantic Web, where reasoning on Description Logic(DL) plays a critical role, it is worth looking into explanation of DL reasoning.

There are three aspects in explanation that we are interested in:

**Understandability and conciseness of explanation**
Explanation means to bring understanding to the human being. Therefore generated explanation has to be easy to understand and reader friendly. It must not be clumsy and lengthened; otherwise it would be very difficult to consume it. Unfortunately, the tableaux algorithm and its optimizations implemented by DL reasoners are complicated in nature, difficult to be understood. Special explanation methodology is needed to generate understandable explanations for DL reasoning.

**Extension of explanation for expressive DLs**
Today tableau-based reasoners like FaCT and RACER are capable of reasoning with the very expressive DL ($\mathcal{SHIQ}$). However, explanation methodologies proposed so far can only handle basic DLs like $\mathcal{ALC}$. Growth of explanation power is far behind that of the reasoning capability. Extending explanation power is therefore one of the goals of our research.

**Integration of explanation facility to DL reasoners**
An explanation engine can not work alone, it must work closely with a DL reasoner. However, current DL reasoners do not support explanation. Modifying an existing DL reasoner to incorporate explanation capability is difficult and costly. A Dual-Reasoner architecture is being investigated to bring reasoning and explanation together.

# Finite Satisfiability of UML class diagrams by Constraint Programming

Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
tmancini@dis.uniroma1.it
www.dis.uniroma1.it/~tmancini

The Unified Modelling Language (UML, cf. www.uml.org) is probably the most used modelling language in the context of software development, and has been proven to be very effective for the analysis and design phases of the software life cycle.

UML offers a number of diagrams for representing various aspects of the requirements for a software application. Probably the most important diagram is the *class diagram*, which represents all main structural aspects of an application: *classes*, *associations*, *ISA hierarchies* between classes, *multiplicity constraints* on associations. Actually, a UML class diagram represents also other aspects, e.g., the attributes and the operations of a class, the attributes of an association, and the specialization of an association. Such aspects, for the sake of simplicity, will not be considered.

A class diagram induces restrictions on the number of objects, because of its multiplicity constraints. In some cases the number of objects of a class is forced to be zero. When a class is forced to have either zero or infinitely many instances, it is said to be *finitely inconsistent* or *finitely unsatisfiable*.

Unsatisfiability, either finite or unrestricted, of a class is a symptom of a bug in the analysis phase, since such a class is clearly superfluous. In particular, finite unsatisfiability is especially relevant in the context of applications, e.g., databases, in which the number of instances is intrinsically finite.

Finite inconsistency may arise in complex situations, and discovering finite inconsistency may be not doable by hand, since it requires global reasoning on the whole class diagram. Thus, automated finite model reasoning in UML class diagrams (including checking finite satisfiability of classes) is of crucial importance for assessing quality of the analysis phase in software development. Indeed, for the purpose of software engineering, finite model reasoning in UML class diagrams is often considered more important than unrestricted reasoning.

In our research we address the implementation of finite model reasoning on UML class diagrams, a task that has not been attempted so far. This is done by exploiting an encoding of UML class diagrams in terms of Description Logics (DLs).

Reasoning in such logics has been studied extensively, both from a theoretical point of view, establishing EXPTIME-completeness of various DL variants, and from a practical point of view, developing practical reasoning systems. The correspondence between UML class diagrams and DLs allows one to use the current state-of-the-art DL reasoning systems to reason on UML class diagrams. However, the kind of reasoning that such systems support is unrestricted (as in first-order logic), and not finite model reasoning. That is, the fact that models (i.e., instantiations of the UML class diagram) must be finite is not taken into account.

Interestingly, in DLs, finite model reasoning has been studied from a theoretical perspective, and its computational complexity has been characterized for various cases. However, no implementations of such techniques have been attempted till now. In our research we reconsider such work, and on the basis of it we present, to the best of our knowledge, the first implementation of finite model reasoning in UML class diagrams.

The main result of our current reseach is that it is possible to use off-the-shelf tools for constraint modelling and programming for obtaining a finite model reasoner. In particular, we propose an encoding of UML class diagram satisfiability as a Constraint Satisfaction Problem (CSP). Moreover, we show also how constraint programming can be used to actually return a finite model of the UML class diagram.

We built a system that accepts as input a class diagram written in the MOF syntax, and translates it into a file suitable for ILOG's OPLStudio, which checks satisfiability and returns a finite model, if there is one. The system allowed us to test the technique on the industrial knowledge base CIM, obtaining encouraging results.

# Pellet: An OWL DL Reasoner

Evren Sirin and Bijan Parsia
MINDSWAP Research Group
University of Maryland, College Park, MD
evren@cs.umd.edu, bparsia@isr.umd.edu

## 1 Introduction

In order to gain experience with description logic reasoner, and to contribute to the OWL Candidate Recommendation process, a small team at MINDSWAP set out to implement a tableau reasoner for the Lite and DL dialects of OWL (corresponding roughly to the description logics SHIF(D) and SHION(D)). Our group found existing, available DL reasoners lacking for our purposes, because we needed an open-source tool that provides ABox reasoning, that does not make Unique Name assumption, supports entailment checks and works with XML Schema datatypes. Pellet has been developed to addresses these issues and has become both our test bed for experiments with DL and Semantic Web reasoning, as well as our standard reasoning component. While not (yet) in the performance range of Racer or Fact, it has many usability features that makes it a good choice for various lighter weight situations.

Technically, Pellet is a sound and complete tableau reasoners for SHIN(D) and SHON(D) (with ABoxes), and a sound but incomplete tableau reasoner for SHION(D) (with ABoxes). Pellet has the usual suite of optimizations including lazy unfolding, absorption, dependency directed backjumping, and semantic branching. It incorporates datatype reasoning for the built-in primitive XML Schema datatypes. Pellet is implemented in pure Java and available as open source software.

## 2 Special features

Pellet has a number of features either driven by OWL requirements or Semantic Web issues.

**Ontology analysis and repair** OWL has two major dialects, OWL DL and OWL Full, with OWL DL being a subset of OWL Full. All OWL knowledge bases are encoded as RDF/XML graphs. OWL DL imposes a number of restrictions on RDF graphs, some of which are substantial (e.g., that the set of class names and individual names be disjoint) and some less so (that every item have a "type" triple). Ensuring that an RDF/XML document meets all the restrictions is a relatively difficult task for authors, and many existing OWL documents are nominally OWL Full, even though their authors intend for them to be OWL DL. Pellet incorporates a number of heuristics to detect "DLizable" OWL Full documents "repair" them.

**Datatype reasoning** XML Schema has a rich set of basic datatypes including various numeric types (integers and floats), strings, and date/type types. It also has several mechanism, both standard and unusual for creating new types out of the base types. For example, it's possible to define a datatype by restricting the integers to the set of integers whose canonical representation has only 10 digits, or whose string representation matches a certain regular expression. Currently, XML Schema systems tend toward validation of documents and generation of PSVI instead of type checking (though, with the advent of XQuery, this might change). Pellet can test the satisfiability of conjunctions of thus constructed datatypes.

**Entailment** In Semantic Web, entailment is the key inference whereas the Description Logic community have focused on satisfiability and subsumption. While entailment can be reduced to satisfiability, most DL systems do not support it. In part to pass a large portion of the OWL test suite, we implemented entailment support in Pellet.

**Conjunctive ABox query** Query answering is yet another important feature for Semantic Web. We have implemented an ABox query answering module in Pellet using "rolling-up" technique. We have devised algorithms to optimize the query answering by changing how likely candidates for variables are found and tried. Exploiting the dependencies between different variable bindings helps us to reduce the total number of satisfiability tests thus speeding up the answer significantly.

## 3 Applications and Future Work

We expose most of Pellet's capabilities equally from a Java API, a command line interface, and a Web form. The Web form has been used by a number of people for species validation, consistency checking, and experimenting with OWL DL classification and entailment.

Pellet is the default reasoner in SwoopEd, a lightweight ontology browser and editor. Pellet is used for classification, class satisfiability testing, query, and species validation and repair.

We also use Pellet for web service discovery and composition. Pellet has been incorporated as the knowledge base for a version of the SHOP2 HTN planning system and Fujitsu Lab of America's Task Computing Environment (TCE).

We have begun experimenting with various multi-ontology/logic formalism, such as distributed description logic and E-Connections implementation techniques. Initial results have been both instructive and promising. We also aim to integrate the reasoner with rules to support Semantic Web Rule Language (SWRL). Other future work projects include generating explanations for concept satisfiability, querying using **K** operator and experimenting with non-monotonic reasoning using annotated logics.

# DL Requirements from Medicine and Biology

Stefan Schulz, Department of Medical Informatics
Freiburg University Hospital, Germany
stschulz@uni-freiburg.de

The need for semantically precise domain descriptions has given raise to an increasing number of so-called bio-ontologies covering different fields of biology and medicine. Examples are the Foundational Model of Anatomy (FMA)[5], the Gene Ontology [3] and the Open Biological Ontologies [6]. There are several examples of the conversion of biomedical ontologies into T-Boxes [2, 7, 1, 8] in order to enable terminological reasoning. The biomedical domain, however, exhibits certain peculiarities. Besides the generally large size of biomedical terminology systems (10,000 - 100,000 concepts) the physical composition of organisms (anatomy, biological structure) plays an pivotal role which which may impact the performance of description logics implementations:

- Part-Whole hierarchies constitute an important ordering principle.
- Definitory cycles are common, e.g. $Cell \sqsubseteq \exists haspart.Cytoplasm$ and $Cytoplasm \sqsubseteq \exists partof.Cell$.
- Pairwise disjunction in taxonomies, e.g. $Organ \sqsubseteq \neg Tissue$
- Pairwise disjunction in partonomies, e.g. $\exists partof.Trunk \sqsubseteq \neg \exists partof.Head$
- Role inclusion, e.g. $r \circ s \sqsubseteq r$ [4]

## References

[1] R. Beck and S. Schulz. Logic-based remodeling of the digital anatomist foundational model. *AMIA 2003 – Proc. of the 2003 Symposium of the American Medical Informatics Association*, pages 687–691, 2003.

[2] A. Gangemi, D. M. Pisanelli, and G. Steve. An overview of the ONION project: Applying ontologies to the integration of medical terminologies. *Data & Knowledge Engineering*, 31(2):183–220, 1999.

[3] Gene Ontology Consortium. Creating the Gene Ontology resource: Design and implementation. *Genome Research*, 11(8):1425–1433, 2001.

[4] I. Horrocks and U. Sattler. The effect of adding complex role inclusion axioms in description logics. In *IJCAI'03 – Proc. of the 18th International Joint Conference on Artificial Intelligence*, pages 343–348, 2003.

[5] C. Rosse and J. L. V. Mejino. A reference ontology for bioinformatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics*, 36:478–500, 2003.

[6] OBO. *Open Biological Ontologies (OBO)*. http://obo.sourceforge.net/, 2004.

[7] S. Schulz and U. Hahn. Medical knowledge reengineering – converting major portions of the UMLS into a terminological knowledge base. *International Journal of Medical Informatics*, 64(2/3):207–221, 2001.

[8] C. J. Wroe, R. Stevens, C. A. Goble, and M. Ashburner. A methodology to migrate the gene ontology to a description logic environment using DAML+OIL. *PSB 2003 – Proc. of the Pacific Symposium on Biocomputing 2003*, pages 624–635, 2003.