Pascal Hitzler
Thomas Lukasiewicz (Eds.)

LNCS 6333

Web Reasoning and Rule Systems

Fourth International Conference, RR 2010
Bressanone/Brixen, Italy, September 2010
Proceedings

RR2010

Springer

Pascal Hitzler   Thomas Lukasiewicz (Eds.)

Web Reasoning and Rule Systems

Fourth International Conference, RR 2010
Bressanone/Brixen, Italy, September 22-24, 2010
Proceedings

Springer

Volume Editors

Pascal Hitzler
Wright State University
Dept. of Computer Science and Engineering
3640 Colonel Glenn Hwy, Dayton, OH 45435, USA
E-mail: pascal.hitzler@wright.edu

Thomas Lukasiewicz
Oxford University
Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
E-mail: thomas.lukasiewicz@comlab.ox.ac.uk

Library of Congress Control Number: 2010934297

CR Subject Classification (1998): D.4, H.2.4, I.2.4, H.3.5, H.5, D.1.6

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN	0302-9743
ISBN-10	3-642-15917-6 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-15917-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The Semantic Web aims at allowing knowledge to be freely accessed and exchanged by software. It is now widely recognized that if the Semantic Web is to contain deep knowledge, the need for new representation and reasoning techniques is critical. These techniques need to find the right trade-off between expressiveness, scalability, and robustness to deal with the inherently incomplete, contradictory, and uncertain nature of knowledge on the Web.

The annual International Conference on Web Reasoning and Rule Systems (RR) addresses these needs and has grown into a major international forum for the discussion and dissemination of new results concerning Web Reasoning and Rule Systems. The first three International Conferences on Web Reasoning and Rule Systems (see http://www.rr-conference.org), held in Innsbruck, Austria (2007), Karlsruhe, Germany (2008), and Chantilly, Virginia, USA (2009), received enthusiastic support from the Web Reasoning community.

This volume contains the papers presented at the Fourth International Conference on Web Reasoning and Rule Systems (RR 2010), which was held in Bressanone/Brixen, Italy, September 22-24, 2010, and which continued the excellence of the RR series. It contains nine full papers, six short papers, four poster/position papers, one PhD paper, and two system descriptions, which were selected out of 31 submissions following a rigorous reviewing process, where each submission was reviewed by at least three program committee members. The volume also contains extended abstracts of the three invited talks/tutorials.

We wish to thank all authors who submitted papers and all conference participants for fruitful discussions. We are grateful to Georg Gottlob and Evren Sirin for their invited talks and Axel Polleres for his tutorial at the conference. We would like to thank the program committee members and additional reviewers for their timely expertise in carefully reviewing the submissions. Special thanks to Diego Calvanese and Mariano Rodríguez-Muro from the Free University of Bozen-Bolzano for the organization of the conference and wonderful days in Bressanone/Brixen. We would like to thank José Júlio Alferes for acting as general chair, Krzysztof Janowicz for his work as sponsorship chair, and all sponsors for their financial support. Many thanks also to the developers of the EasyChair conference system, which we used for the reviewing process and the preparation of this volume.

September 2010

Pascal Hitzler
Thomas Lukasiewicz

Conference Organization

RR 2010 was organized by the Free University of Bozen-Bolzano.

General Chair

José Júlio Alferes	Universidade Nova de Lisboa, Portugal

Program Chairs

Pascal Hitzler	Wright State University, USA
Thomas Lukasiewicz	Oxford University, UK

Local Arrangements

Diego Calvanese	Free University of Bozen-Bolzano, Italy
Mariano Rodríguez-Muro	Free University of Bozen-Bolzano, Italy

Sponsorship Chair

Krzysztof Janowicz	Pennsylvania State University, USA

Program Committee

Grigoris Antoniou	University of Crete, Greece
Marcelo Arenas	PUC Chile, Chile
Jie Bao	Rensselaer Polytechnic Institute, USA
Chitta Baral	Arizona State University, USA
Leopoldo Bertossi	Carleton University, Canada
Andrea Calì	Oxford University, UK
Vinay Chaudri	Stanford Research Institute, USA
Kendall Clark	Clark & Parsia, USA
Claudia d'Amato	University of Bari, Italy
Carlos Damasio	Universidade Nova de Lisboa, Portugal
Wlodek Drabent	IPI PAN Warszawa, Poland
Sergio Flesca	University of Calabria, Italy
Christine Golbreich	University of Versailles Saint-Quentin, France
Claudio Gutierrez	University of Chile, Chile
Stijn Heymans	TU Vienna, Austria
Rinke Hoekstra	VU Amsterdam, The Netherlands

Additional Reviewers

Giorgos Flouris	Bruno Marnette	Giorgio Orsi
Jorge Pérez	Pierfrancesco Veltri	Xiaowang Zhang

Sponsors

We thank the Free University of Bozen-Bolzano for their very generous and substantial financial support which made this event possible.

FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN · BOLZANO

This conference was partially supported by the Platinum Sponsor DERI as main conference sponsor, the Gold Sponsors Clark & Parsia, Ontotext, and Vulcan, the Silver Sponsors SimCat and LarKC, and the Regular Sponsors IOS Press, CRC Press, 52° North, and GeoVISTA.

Platinum Sponsor

Gold Sponsors

Table of Contents

Query Answering under Non-guarded Rules in Datalog+/-

Andrea Calì[3,2], Georg Gottlob[1,2], and Andreas Pieris[1]

[1] Computing Laboratory, University of Oxford, UK
[2] Oxford-Man Institute of Quantitative Finance, University of Oxford, UK
[3] Department of Information Systems and Computing, Brunel University, UK
firstname.lastname@comlab.ox.ac.uk

Abstract. In ontology-based data access, an extensional database is enhanced by an ontology that generates new intensional knowledge which has to be considered when answering queries. In this setting, tractable *data complexity* (i.e., complexity w.r.t. the data only) of query answering is crucial, given the need to deal with large data sets. A well-known class of tractable ontology languages is the DL-lite family; however, in DL-lite it is impossible to express simple and useful integrity constraints that involve joins. To overcome this limitation, the Datalog+/- class of decidable languages uses *tuple-generating dependencies (TGDs)* as rules, thus allowing for conjunctions of atoms in the rule bodies, with suitable limitations to ensure decidability. In particular, *sticky* sets of TGDs allow for joins and variable repetition in rule bodies under certain conditions. In this paper we extend the notion of stickiness by introducing *weakly-sticky* sets of TGDs, which also generalize the well-known weakly-acyclic sets of TGDs. We investigate the complexity of query answering under such language, and in addition we provide novel complexity results on weakly-acyclic sets of TGDs. Moreover, we present the novel class of *sticky-join* sets of TGDs, which generalizes both sticky sets of TGDs and so-called linear TGDs, an extension of inclusion dependencies.

1 Introduction

Due to the complex knowledge representation needs in today's information systems, traditional database systems are being enhanced with advanced reasoning and query processing features. In the business world, enterprise data reside usually at relational databases, and complex constraints have to be enforced. Both the knowledge representation and the database communities agree upon the need for ontological features on top of the raw data.

In ontology-enhanced database systems, an extensional relational database D (also referred-to as ABox in the description logic community) is combined with an *ontological theory* Σ (also called TBox) describing rules and constraints which derive new intensional data from the extensional data. Queries are then to be answered on the whole theory $D \cup \Sigma$ rather than on D alone: given a Boolean conjunctive query q, we check whether $D \cup \Sigma \models q$. Similarly, if $q(\mathbf{X}) =$

$\exists \mathbf{Y}\, body(\mathbf{X}, \mathbf{Y})$ is a conjunctive query with output variables $\mathbf{X}$, then its answer in the ontological database consists of all tuples $\mathbf{t}$ of constants such that $D \cup \Sigma \models \exists \mathbf{u}\, body(\mathbf{t}, \mathbf{u})$. Interestingly, the well-known notion of *chase* [17,11,16,14] of a database D according to an ontology Σ, denoted $chase(D, \Sigma)$, is a useful tool for query answering. It can be shown that the answers to a query q against $D \cup \Sigma$ coincide with the answers to q evaluated over $chase(D, \Sigma)$ [17,11,16,14].

A crucial issue is that the chase expansion does not terminate in general, and this happens in a large number of real-world cases, even when the ontology consists of inclusion dependencies only. A milestone work that deals with non-terminating chase is the one by Johnson and Klug [17], where it is shown that, under inclusion dependencies (plus a certain class of key dependencies), an initial, finite portion of the chase is sufficient for query answering. However, inclusion dependencies alone, or with the addition of key dependencies, are not sufficiently expressive to capture the most common ontological constraints. The goal is therefore to achieve good expressive power while retaining efficiency of query answering in terms of *data complexity*, i.e., the complexity calculated having D only as input, while q and Σ are considered to be fixed.

A significant contribution in this direction was the introduction of the DL-lite family of description logics [12,24]. DL-lite languages enjoy the desirable property of being *first-order rewritable*, henceforth abbreviated as *FO-rewritable*, i.e., for every query q against $D \cup \Sigma$, it is possible to compute a *first-order* query q_Σ based on Σ, such that evaluating q against $D \cup \Sigma$ returns the same answers as evaluating q_Σ directly against D. The latter evaluation can be done in the low complexity class AC_0 in data complexity; moreover, since every first-order query can be encoded in SQL, it can be passed to a relational DBMS and therefore being executed efficiently and with all the DBMS's optimizations.

A more expressive class of languages is the Datalog$^\pm$ family, whose rules are *tuple-generating dependencies (TGDs)*, with suitable restrictions that ensure decidability of query answering. TGDs are analogous to Datalog rules with value invention (see, e.g., [21,4]); this extension of the well-known Datalog language [1] allows us to overcome the limitations of Datalog for ontology modeling that are pointed out, for instance, in [23]. More specifically, *linear Datalog*$^\pm$ [7] (linear TGDs) properly generalizes the DL-lite family with the addition (which does not increase the complexity of query answering) of negative constraints and key dependencies. In linear Datalog$^\pm$ rules are TGDs with exactly one atom in the body. Linear Datalog$^\pm$ is a sub-formalism of *guarded Datalog*$^\pm$ [7,5] (guarded TGDs), which is in turn generalized by *weakly-guarded Datalog*$^\pm$ [5] (weakly-guarded sets of TGDs). We refer the reader to the above cited literature for more details on such Datalog$^\pm$ languages. Unfortunately, none of the above formalisms is expressive enough to be able to model simple real-life cases such as the one in the following example, taken from [8].

Example 1. Consider the following relational schema, which shall be used as our running example.

$$\begin{array}{l} dept(\mathsf{Dept\_Id}, \mathsf{Mgr\_Id}), \\ emp(\mathsf{Emp\_Id}, \mathsf{Dept\_Id}, \mathsf{Area}, \mathsf{Project\_Id}), \end{array}$$

$$\begin{array}{l} runs(\mathsf{Dept\_Id}, \mathsf{Project\_Id}), \\ in\_area(\mathsf{Project\_Id}, \mathsf{Area}), \\ project\_mgr(\mathsf{Emp\_Id}, \mathsf{Project\_Id}), \\ external(\mathsf{Ext\_Id}, \mathsf{Area}, \mathsf{Project\_Id}). \end{array}$$

The fact that each department has an employee as manager can be expressed by the TGD

$$dept(V, W) \rightarrow \exists X \exists Y \exists Z\, emp(W, X, Y, Z).$$

The following TGD expresses the fact that each employee works on some project that falls into his/her area of specialization ran by his/her department.

$$emp(V, W, X, Y) \rightarrow \exists Z\, dept(W, Z), runs(W, Y), in\_area(Y, X).$$

The TGD below states that for each project run by some department there exists an external controller, specialized on the area of the project, that works on it.

$$runs(W, X), in\_area(X, Y) \rightarrow \exists Z\, external(Z, Y, X).$$

Notice first that the above TGDs do not guarantee the termination of the chase for every initial database. Moreover, the third TGD contains a join (over variable X) which is precisely of the kind that cannot be represented in guarded Datalog$^{\pm}$, let alone in DL-lite. ■

While query answering under TGDs is undecidable [3,5], we observed that such undecidability is due to awkward cases that rarely occur in practice. This led us to introducing a novel paradigm called *stickiness*. We introduced in [8] the tractable class of *sticky* sets of TGDs, which allows for (a slightly restricted form of) joins in rule bodies while being at the same time FO-rewritable. The formal definition of sticky sets of TGDs is given in Subsection 2.3. Sticky sets of TGDs impose a mild limitation on multiple occurrences of variables in the body of TGDs, and they are useful in practice. However, it is important to notice that they do not generalize the well-known class of weakly-acyclic TGDs [16], which is at the basis of the data exchange framework, and which ensures chase termination for every instance. There are indeed cases of weakly-acyclic sets of TGDs that are not sticky, while they are of course decidable (though not FO-rewritable in general). In this paper we go beyond stickiness in order to capture also the expressive power of weakly-acyclic sets of TGDs, and more.

Summary of contributions. In Section 3 we first consider weakly-acyclic sets of TGDs. We show that query answering under weakly-acyclic sets of TGDs is 2EXPTIME-hard in combined complexity (i.e., the complexity considering the database, the ontology and the query as input). Then, we consider the existence-of-solution problem in data exchange [16,19], in the presence of weakly-acyclic sets of TGDs and also of *equality-generating dependencies (EGDs)*. EGDs, for which we refer the reader to [16,8], are assertions of the form $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow X_i = X_j$, where X_i and X_j are variables of $\mathbf{X}$. We prove that the existence-of-solution problem is 2EXPTIME-hard in combined complexity. In Section 4 we introduce the novel class of *weakly-sticky* sets of TGDs. We establish precise complexity

	Data Complexity	**Combined Complexity**
STGDs BCQ answering	FO-rewritable [8]	EXPTIME-complete LB: [8], UB: [8]
WATGDs BCQ answering	PTIME-complete LB: [13], UB: [16]	2EXPTIME-complete LB: Thm. 1, UB: [16,22]
WATGDs + EGDs existence-of-solution	PTIME-complete LB: [19], UB: [19]	2EXPTIME-complete LB: Thm. 2,[18], UB: [16,22]
WSTGDs BCQ answering	PTIME-complete LB: [13], UB: Thm. 4	2EXPTIME-complete LB: Thm. 1, UB: Thm. 3
SJTGDs BCQ answering	FO-rewritable Thm. 5	EXPTIME-complete LB: [8], UB: Thm. 5
WSJTGDs BCQ answering	PTIME-complete LB: [13], UB: Thm. 6	2EXPTIME-complete LB: Thm. 1, UB: Thm. 6

Fig. 1. Summary of complexity results

bounds for the problem of query answering under weakly-sticky sets of TGDs, both in combined and data complexity. Finally, in Section 5 we present several extensions of sticky sets of TGDs. In particular, we present the novel language of *sticky-join* Datalog$^\pm$ (sticky-join sets of TGDs) that extends both sticky sets of TGDs and linear TGDs, while keeping FO-rewritability. The extension of sticky Datalog$^\pm$ to capture linear Datalog$^\pm$ is natural, as there exist simple sets of linear TGDs which are non-sticky[1]: we investigated why the variable repetitions in such sets do not cause undecidability of query answering, and this led us to discover sticky-join Datalog$^\pm$ and to prove the related complexity results. Also, we introduce the novel class of *weakly-sticky-join* sets of TGDs that extends both weakly-acyclic and sticky-join sets of TGDs.

Figure 1 summarizes our complexity results. STGDs, WATGDs, WSTGDs, SJTGDs and WSJTGDs are abbreviations for sticky, weakly-acyclic, weakly-sticky, sticky-join and weakly-sticky-join sets of TGDs, respectively. UB and LB stand for upper and lower bound, respectively. Notice that all our complexity results, derived for Boolean conjunctive queries, carry over, as usual, to the (decision) query answering problem for general (non-Boolean) conjunctive queries (see, e.g., [5]), as well as to the conjunctive query containment problem. For most proofs of results in this paper, additional details, and possible corrections, we refer the reader to a report available online [9]. For details on sticky-sets of TGDs see [8] (which is preliminary to the present paper). For an overview on some relevant Datalog$^\pm$ languages, we invite the reader to consult our work [10].

2 Background

2.1 Technical Definitions

In this subsection we recall some basics on databases, queries, TGDs, and the TGD chase procedure.

[1] Notice that, on the contrary, sets of inclusion dependencies are always sticky.

General. We define the following pairwise disjoint (infinite) sets of symbols: a set Γ of *constants* (constitute the "normal" domain of a database), and a set Γ_N of *labeled nulls* (used as placeholders for unknown values, and thus can be also seen as variables). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. A lexicographic order is defined on $\Gamma \cup \Gamma_N$, such that every value in Γ_N follows all those in Γ.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. We write r/n to denote that the predicate r has arity n. A *position* $r[i]$ (in a schema $\mathcal{R}$) is identified by a predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where r/n is a relation, and $t_1, \ldots, t_n$ are terms. For an atom $\underline{a}$, we denote as $dom(\underline{a})$ and $var(\underline{a})$ the set of its terms and the set of its variables, respectively. These notations naturally extends to sets and conjunctions of atoms. Conjunctions of atoms are often identified with the sets of their atoms.

A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \to S_2$ defined as follows: *(i)* $\varnothing$ is a substitution (empty substitution), *(ii)* if h is a substitution, then $h \cup \{X \to Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$, and h does not already contain some $X \to Z$ with $Y \neq Z$. If $X \to Y \in h$, then we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema $\mathcal{R}$, is a substitution $h : dom(A_1) \to dom(A_2)$ such that: *(i)* if $t \in \Gamma$, then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n)$ is in A_1, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2. The notion of homomorphism naturally extends to conjunctions of atoms.

Databases and Queries. A *database (instance)* D for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$ (a.k.a. *facts*), where $r/n \in \mathcal{R}$ and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. We denote as $r(D)$ the set $\{\mathbf{t} \mid r(\mathbf{t}) \in D\}$. A *conjunctive query (CQ)* q of arity n over a schema $\mathcal{R}$, written as q/n, has the form $q(\mathbf{X}) = \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, $\mathbf{X}$ and $\mathbf{Y}$ are sequences of variables or constants in Γ, and the length of $\mathbf{X}$ is n. $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as $body(q)$. A *Boolean CQ (BCQ)* is a CQ of zero arity. The *answer* to a CQ q/n over a database D, denoted as $q(D)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Gamma \cup \Gamma_N$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer, in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over D, denoted as $D \models q$, iff $\langle\rangle \in q(D)$.

Tuple-Generating Dependencies. A *tuple-generating dependency (TGD)* σ over a schema $\mathcal{R}$ is a first-order formula $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted as $body(\sigma)$ and $head(\sigma)$, respectively. Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs. Such σ is satisfied by a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq D$.

We now define the notion of *query answering* under TGDs. Given a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$, is the set of all databases B such that $B \models D \cup \Sigma$, which means that $B \supseteq D$ and B satisfies Σ. The *answer* to a CQ q w.r.t. D and Σ, denoted as $ans(q, D, \Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in q(B)$ for each $B \in mods(D, \Sigma)\}$. The *answer* to a BCQ q w.r.t. D and Σ is positive, denoted as $D \cup \Sigma \models q$, iff $ans(q, D, \Sigma) \neq \varnothing$. Note that query answering under general TGDs is undecidable [3], even when the schema and the set of TGDs are fixed [5].

We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent. Moreover, it is easy to see that the query output tuple problem (as a decision version of CQ evaluation) and BCQ evaluation are AC$_0$-reducible to each other. Henceforth, we thus focus only on the BCQ evaluation problem. All complexity results carry over to the other problems.

The TGD Chase. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [20], and later for checking query containment [17]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted database satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD *chase rule*. The TGD chase rule comes in two equivalent fashions: *oblivious* and *restricted* [5], where the restricted one repairs TGDs only when they are not satisfied. In the sequel, we focus on the oblivious one for technical clarity. The TGD chase rule defined below is the building block of the chase.

TGD CHASE RULE: Consider a database D for a schema $\mathcal{R}$, and a TGD $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. If σ is *applicable* to D, i.e., there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ then: *(i)* define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \mathbf{Z}$, where $z_i \in \Gamma_N$ is a "fresh" labeled null not introduced before, and following lexicographically all those introduced so far, and *(ii)* add to D the set of atoms in $h'(\psi(\mathbf{X}, \mathbf{Z}))$ if not already in D.

Given a database D and a set of TGDs Σ, the chase algorithm for D and Σ consists of an exhaustive application of the TGD chase rule in a breadth-first fashion, which leads as result to a (possibly infinite) chase for D and Σ, denoted as $chase(D, \Sigma)$. For the formal definition of the chase algorithm we refer the reader to [6]. The (possibly infinite) chase for D and Σ is a *universal model* of D w.r.t. Σ, i.e., for each database $B \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to B [16,14]. Using this fact it can be shown that for a BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$.

2.2 Weakly-Acyclic Sets of TGDs: Preliminaries

In this subsection we present the class of *weakly-acyclic* sets of TGDs [15,16]. We first recall the notion of the dependency graph as defined in [16].

Definition 1. *Given a set Σ of TGDs over a schema $\mathcal{R}$, the* dependency graph *of Σ is the directed graph constructed as follows. There exists a node for each position $r[i]$ in $\mathcal{R}$, where $r/n \in \mathcal{R}$ and $i \in \{1, \ldots, n\}$. For each TGD $\sigma \in \Sigma$, for* *each $\forall$–variable V in head(σ), and for each occurrence of V in body(σ) at position $r[i]$, apply the following two steps: (i) for each occurrence of V in head(σ) at position $s[j]$, add an arc from $r[i]$ to $s[j]$ (if it does not already exist), and (ii) for each $\exists$–variable W, and for each occurrence of W in head(σ) at position $t[k]$, add a* special *arc from $r[i]$ to $t[k]$ (if it does not already exist). The* rank *of a position $r[i]$ is the maximum number of special arcs over all (finite or infinite) paths ending at $r[i]$ in the dependency graph of Σ.*

Given a set Σ of TGDs over a schema $\mathcal{R}$, the set of positions in $\mathcal{R}$ can be partitioned into sets of positions with finite and infinite rank, denoted as $\Pi_F(\mathcal{R}, \Sigma)$ and $\Pi_\infty(\mathcal{R}, \Sigma)$, respectively. When $\mathcal{R}$ and Σ are obvious from the context, we shall denote the above sets with Π_F and Π_∞, respectively. Intuitively, Π_F (resp., Π_∞) is the set of position where a finite (resp., infinite) number of distinct values can appear during the construction of the chase. We are now ready to give the definition of weakly-acyclic sets of TGDs [15,16].

Definition 2. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. Σ is* weakly-acyclic *iff $\Pi_F = \mathcal{R}$ and $\Pi_\infty = \varnothing$.*

Example 2. Let $\mathcal{R}$ be the relational schema given in Example 1. Consider the set Σ of TGDs over $\mathcal{R}$ constituted by the following TGDs:

$$\begin{aligned} emp(V, W, X, Y) &\rightarrow \exists Z\, dept(W, Z), runs(W, Y), \\ runs(W, X), dept(W, Y) &\rightarrow project\_mgr(Y, X). \end{aligned}$$

It is not difficult to verify that $\Pi_F = \mathcal{R}$ and $\Pi_\infty = \varnothing$, which implies that Σ is weakly-acyclic. ∎

BCQ answering under weakly-acyclic sets of TGDs is in 2EXPTIME in combined complexity. This is implied from results in [16,22]. In particular, given a weakly-acyclic set Σ of TGDs over a schema $\mathcal{R}$, during the construction of the chase only double-exponentially many distinct values can appear in any position of $\mathcal{R}$, and thus only double-exponentially many tuples can occur in the chase. Therefore, given a database D for $\mathcal{R}$, and a BCQ q over $\mathcal{R}$, we just evaluate q over the *finite* chase of D and Σ, which can be constructed in 2EXPTIME. The lower bound is studied in Section 3.

2.3 Sticky Sets of TGDs: Preliminaries

We now recall the class of *sticky* sets of TGDs introduced in [8]. Stickiness, formally defined below by an efficiently testable condition involving variable-marking, is a sufficient syntactic condition than ensures the so-called *sticky property* of the chase, which is as follows. For every instance D, assume that during the chase of D under a set Σ of TGDs, we apply a TGD σ that has a variable V appearing more than once in its body; assume also that V maps (via homomorphism) on the symbol z, and that by virtue of this application the atom $\underline{a}$ is introduced. In this case, for each atom $\underline{b}$ in *body*(σ), we say that $\underline{a}$ is derived from $\underline{b}$. Then, we have that z appears in $\underline{a}$, and in all atoms resulting from some chase derivation sequence starting from $\underline{a}$, "sticking" to them (hence the name "sticky sets of TGDs").

In [8], for technical reasons, are considered constant-free TGDs, where each head-variable occurs just once in a certain head-atom. In that case stickiness is also a necessary condition for the sticky property of the chase. In the present paper we consider arbitrary TGDs without the assumptions of [8]. We now come to the formal definition of sticky sets of TGDs.

Definition 3. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. We mark the variables that occur in the body of the TGDs of Σ according to the following marking procedure. First, for each TGD $\sigma \in \Sigma$ and for each variable V in body(σ), if there exists an atom $\underline{a}$ in head(σ) such that V does not appear in $\underline{a}$, then we mark each occurrence of V in body(σ). Now, we apply exhaustively (i.e., until a fixpoint is reached) the following step: for each TGD $\sigma \in \Sigma$, if a marked variable in body(σ) appears at position π, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma' = \sigma$), we mark each occurrence of the variables in body(σ') that appear in head(σ') at the same position π. We say that Σ is* sticky *iff there is no TGD $\sigma \in \Sigma$ such that a marked variable occurs in body(σ) more than once.*

Example 3. Let Σ be the set of TGDs given in Example 1. According to the marking procedure in Definition 3, we mark the variables as follows (we mark variables with a cap, e.g., $\hat{X}$):

$$\begin{aligned} dept(\hat{V},\hat{W}) &\rightarrow \exists X \exists Y \exists Z\, emp(W,X,Y,Z), \\ emp(\hat{V},\hat{W},\hat{X},\hat{Y}) &\rightarrow \exists Z\, dept(W,Z), runs(W,Y), in\_area(Y,X), \\ runs(\hat{W},X), in\_area(X,Y) &\rightarrow \exists Z\, external(Z,Y,X). \end{aligned}$$

Clearly, for each TGD $\sigma \in \Sigma$, there is no marked variable that occurs in *body*(σ) more than once. Therefore, Σ is a sticky set of TGDs. It is easy to verify that Σ is not weakly-acyclic. In fact, the classes of weakly-acyclic and sticky sets of TGDs, presented in this section, are incomparable. ∎

Interestingly, stickiness is a sufficient property that ensures that the TGDs are a so-called *finite unification set*, an abstract decidability paradigm defined in [2]. BCQ answering under sticky sets of TGDs is EXPTIME-complete in combined complexity [8]. Also, sticky sets are FO-rewritable, which implies that query answering is in AC_0 in data complexity [8]. The lower bound for the combined complexity is proved by showing the EXPTIME-hardness of the fact inference problem for *lossless Datalog*, whose rules are a special case of sticky sets of TGDs. Lossless Datalog programs are a special case of Datalog programs, where each rule enjoys the following property: all variables appearing in the rule body also appear in the rule head. The upper bounds for both the combined and the data complexity are established by exhibiting an alternating algorithm, called Sticky-QAns. In what follows we give a rough description of this algorithm.

Recall that query answering under sticky sets of TGDs is equivalent to query answering under sticky sets of TGDs with just one atom in their heads. This is established by using the LOGSPACE transformation of [5] from sets of TGDs to sets of TGDs with a single head-atom, and observing that such transformation does not alter the stickiness property. In the rest of the paper, unless otherwise stated, we assume w.l.o.g. that every TGD has just one atom in its head.

Algorithm Sticky-QAns. The algorithm takes as input a database D for a schema $\mathcal{R}$, a sticky set Σ of TGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$, and outputs *accept* iff $chase(D, \Sigma) \models q$. The algorithm works as follows:

1. Non-deterministically guess the following: two disjoint sets $S_1 \subseteq var(q)$ and $S_2 \subset var(q)$, and a substitution $h : S_1 \rightarrow S_2 \cup dom(D)$; let $S = var(q) \setminus S_1$.
2. For each variable $V \in S$, non-deterministically guess an atom $\underline{a}_V$ such that: $dom(\underline{a}_V) \subseteq dom(D) \cup S \cup \{\star_1, \ldots, \star_w\}$, where w is the maximum arity over all predicates in $\mathcal{R}$; let $G = \{\langle V, \underline{a}_V \rangle\}_{V \in S}$.
3. If $\mathsf{Proof}(h(body(q)), D, \Sigma, S, G)$ accepts, then *accept*; otherwise, *reject*.

Intuitively, in stage 1 we guess which of the variables in the body of q are mapped onto a constant of $dom(D)$, and which constant, during the evaluation of q over $chase(D, \Sigma)$. Moreover, we guess which of the variables in the body of q are mapped onto the same null of Γ_N. In stage 2, roughly speaking, we guess, for each variable $V \in S$, an atom that represents the equality type of the atom in which z_V is invented during the chase, where z_V is the null onto which V is mapped. The special star variables $\star_1, \ldots, \star_w$ represent placeholders for any term in $dom(D) \cup \Gamma_N$. Finally, in stage 3, by calling the recursive alternating procedure Proof, we verify that the given query is entailed by $chase(D, \Sigma)$. The procedure $\mathsf{Proof}(P, D, \Sigma, S, G)$ works as follows:

1. If $P \subseteq D$, then *accept*.
2. Universally select all the atoms $\underline{a}$ in P that do not occur in D; let $G(\underline{a}) = \{\langle V, \underline{a}_V \rangle \mid \langle V, \underline{a}_V \rangle \in G \text{ and } V \in dom(\underline{a})\}$.
3. Existentially guess a TGD $\sigma \in \Sigma$ such that $head(\sigma)$ and $\underline{a}$ unify; if there is no such a TGD, then *reject*. Let $S = \theta(S)$ and $G(\underline{a}) = \theta(G(\underline{a}))$, where θ is the MGU for $head(\sigma)$ and $\underline{a}$. To avoid undesirable clutter between variables, we replace the variables of σ with new ones that do not occur in Σ, and not introduced so far.
4. If at position π in $\underline{a}$ the variable $V \in S$ occurs, and at the same position π in $head(\sigma)$ an $\exists$-variable of σ occurs, then: if $\underline{a}_V$ cannot be obtained from $\underline{a}$ by replacing the variables that do not occur in $G(\underline{a})$ (if any) with other terms, then *reject*.
5. If a subset of D can be obtained from $\theta(body(\sigma))$ by replacing the variables that do not occur in $G(\underline{a})$ (if any) with constants, then *accept*.
6. Let A be the set of atoms in $\theta(body(\sigma))$ from which cannot be obtained an atom in D by replacing the variables that do not occur in $G(\underline{a})$ with constants. If $\mathsf{Proof}(A, D, \Sigma, S, G)$ accepts, then *accept*.

The above alternating algorithm constructs a so-called *resolution proof-tree*. In fact, a resolution proof-tree describes the non-deterministic choices performed during the execution of the algorithm. The leaf nodes are the atoms in the body of the given query. If the algorithm accepts, which implies that the given query is entailed by the chase, then the root nodes are atoms in the given database. From a resolution proof-tree, by simply renaming variables, we can obtain the part of the chase due to which the given query is entailed. The key fact that allows us to establish soundness and completeness of the algorithm, and also to obtain the desired upper bounds, is that only the variables in the body of the given query, and thus only polynomially many symbols, can appear in different branches of a resolution proof-tree. This holds since the TGDs that we consider are sticky.

As we shall see in Section 4, the algorithm Sticky-QAns can be extended in order to tackle BCQ answering under our generalized version of sticky sets of TGDs, the so-called weakly-sticky, that captures both sticky and weakly-acyclic sets of TGDs. Moreover, as discussed in Section 5, the algorithm Sticky-QAns can be employed for BCQ answering under sticky-join sets of TGDs, a class that captures both sticky sets of TGDs and linear TGDs.

3 Weakly-Acyclic Sets of TGDs

3.1 Combined Complexity: Lower Bound

In this subsection we establish that the lower bound for the combined complexity of BCQ answering under weakly-acyclic sets of TGDs is 2EXPTIME by simulating a 2EXPTIME Turing machine. As already mentioned (see Subsection 2.2), membership in 2EXPTIME is implied from results in [16,22].

Theorem 1. *BCQ answering under weakly-acyclic sets of TGDs is 2*EXPTIME*-hard, even in the case of fixed arity.*

Proof (sketch). Consider a 2EXPTIME Turing Machine (TM) M. We are going to simulate the computation of M on an input string I by constructing a database D, and a weakly-acyclic set Σ of TGDs such that $D \cup \Sigma \models \mathit{accept}$ iff M accepts I, where *accept* is a propositional predicate. Let D be the database constituted by the atoms: $r_0(0)$, $r_0(1)$, $\mathit{succ}_0(0,1)$, $\mathit{min}_0(0)$, $\mathit{max}_0(1)$. For each $i \in \{0,\ldots,n-1\}$, where n is the length of the input string I, we add to Σ the following TGDs:

$$\begin{array}{l}
r_i(X), r_i(Y) \;\rightarrow\; \exists Z\, s_i(X,Y,Z),\\
s_i(X,Y,Z) \;\rightarrow\; r_{i+1}(Z),\\
s_i(X,Y,Z), s_i(X,Y',Z'), \mathit{succ}_i(Y,Y') \;\rightarrow\; \mathit{succ}_{i+1}(Z,Z'),\\
s_i(X,Y,Z), s_i(X',Y',Z'), \mathit{max}_i(Y), \mathit{min}_i(Y'), \mathit{succ}_i(X,X') \;\rightarrow\; \mathit{succ}_{i+1}(Z,Z'),\\
\mathit{min}_i(X), s_i(X,X,Y) \;\rightarrow\; \mathit{min}_{i+1}(Y),\\
\mathit{max}_i(X), s_i(X,X,Y) \;\rightarrow\; \mathit{max}_{i+1}(Y).
\end{array}$$

It is not difficult to show, by induction on i, that in the chase of D and Σ the relation r_n contains 2^{2^n} elements, which are linearly ordered by succ_n. Now that we have a double-exponential number of symbols at hand, we can use them to simulate the double-exponential number of time instants of M; since 2EXPTIME $\subseteq$ 2EXPSPACE (in k steps, a TM cannot write more than k cells), we can use the same symbols to simulate the tape cells of M. From this point on, the reduction (which is omitted due to space reasons) becomes sort of standard, and can be done in the same fashion as in [13]. All the rules encoding the actual behavior of the TM are Datalog rules, which on top of a weakly-acyclic set of TGDs form again a set which is weakly-acyclic. Note that the arity of predicates in the above construction is fixed. $\square$

The following corollary follows immediately from Theorem 1, and the fact that BCQ answering under weakly-acyclic sets of TGDs is in 2EXPTIME [16,22].

Corollary 1. *BCQ answering under weakly-acyclic sets of TGDs in 2*EXPTIME*-complete, even in the case of fixed arity.*

3.2 The Complexity of Data Exchange

Data exchange is the problem of transforming data structured under a schema, called the source schema, to data structured under a different schema, called the target schema [16]. Data exchange has been formalized using the concept of a schema mapping. Formally, a *schema mapping* is a 4-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t \rangle$, where $\mathcal{S}$ is a source schema, $\mathcal{T}$ is a target schema, Σ_{st} is a set of source-to-target (s-t) TGDs, i.e., TGDs where the body is a conjunction over $\mathcal{S}$, and the head is a conjunction over $\mathcal{T}$, and Σ_t is a set of target TGDs and target EGDs, i.e., TGDs and EGDs over $\mathcal{T}$. The *existence-of-solution problem* for $\mathcal{M}$ is the following: given an instance I for $\mathcal{S}$, does a solution for I, that is, a (finite) instance J for $\mathcal{T}$ such that $I \cup J \models \Sigma_{st} \cup \Sigma_t$, exist? [16,19].

By exploiting Theorem 1, we can easily show that, given a schema mapping $\mathcal{M} = \langle \mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t \rangle$, where Σ_t is the union of a weakly-acyclic set of TGDs and a set of EGDs, the existence-of-solution problem for $\mathcal{M}$ is 2EXPTIME-hard. The same result has been established independently by Kolaitis and Panttaja [18] by a reduction from an alternating EXPSPACE Turing machine; however, this result does not imply Theorem 1.

Theorem 2. *Consider a schema mapping $\mathcal{M} = \langle \mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t \rangle$, where Σ_{st} is a set of s-t TGDs, and Σ_t is the union of a weakly-acyclic set of target TGDs and a set of target EGDs. The existence-of-solution problem for $\mathcal{M}$ is 2*EXPTIME*-hard, even in the case of fixed arity.*

Proof (sketch). The proof is by reduction from the BCQ answering problem under weakly-acyclic sets of TGDs. Consider a database D for a schema $\mathcal{R}$, a weakly-acyclic set Σ of TGDs over $\mathcal{R}$, and a BCQ $q = \exists \mathbf{X}\, \varphi(\mathbf{X})$ over $\mathcal{R}$. We construct the schema mapping $\mathcal{M} = \langle \mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t \rangle$ as follows. Let $\mathcal{S} = \{r^\star \mid r \in \mathcal{R}\} \cup \{p^\star\}$ and $\mathcal{T} = \mathcal{R} \cup \{p, s\}$, where $p/1$ and $s/2$ are auxiliary predicates that do not belong in $\mathcal{R}$. The s-t TGDs in Σ_{st} copy the initial instance from $\mathcal{S}$ to $\mathcal{T}$, i.e., for each predicate $r^\star \in \mathcal{S}$ of arity n, we have in Σ_{st} the s-t TGD $r^\star(X_1, \ldots, X_n) \rightarrow r(X_1, \ldots, X_n)$. Let $\Sigma_t = \Sigma \cup \{\sigma, \eta\}$ with σ be the TGD $\varphi(\mathbf{X}), p(U), p(V) \rightarrow s(U, V)$, where U and V are variables that do not occur in

$\mathbf{X}$, and η be the EGD $s(X,Y), s(X,Z) \rightarrow Y = Z$. Clearly, $\Sigma \cup \{\sigma\}$ is weakly-acyclic since there are no $\exists$-variables in σ. Finally, let $I = \{r^\star(\mathbf{t}) \mid r \in \mathcal{R}$ and $\mathbf{t} \in r(D)\} \cup \{p^\star(a), p^\star(b)\}$, where $a, b \in \Gamma$, be the initial instance for $\mathcal{S}$. It is easy to see that $D \cup \Sigma \models q$ iff the answer to the existence-of-solution problem for $\mathcal{M}$, given instance I, is negative. This completes the proof. □

In [19], the existence-of-solution problem for a schema mapping $\mathcal{M}$, where the target dependencies is the union of a weakly-acyclic set of TGDs and a set of EGDs, is stated to be EXPTIME-complete. This bug was fixed in the conference talk of the paper, where membership in 2EXPTIME was showed, while the lower bound was left as an open problem; see also [22]. The next result follows.

Corollary 2. *The existence-of-solution problem for a schema mapping $\mathcal{M}$, where the target dependencies is the union of a weakly-acyclic set of TGDs and a set of EGDs is 2*EXPTIME*-complete, even in the case of fixed arity.*

4 Weakly-Sticky Sets of TGDs

In this section we present a class, called *weakly-sticky* sets of TGDs, that generalizes both weakly-acyclic (and thus Datalog) and sticky sets of TGDs. Roughly, in a weakly-sticky set of TGDs, the variables that occur more than once in the body of a TGD are either non-marked[2], or they occur at positions where a finite number of distinct values can appear during the chase.

Definition 4. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. We say that Σ is* weakly-sticky *iff for each $\sigma \in \Sigma$ and for each variable V that occurs more than once in the body of σ, the following condition holds: V is a non-marked variable, or at least one occurrence of V in $body(\sigma)$ occurs at some position in Π_F.*

Recall that a set Σ of TGDs over a schema $\mathcal{R}$ is weakly-acyclic iff all positions of $\mathcal{R}$ are in Π_F. This immediately implies that every weakly-acyclic set of TGDs is also weakly-sticky. On the other hand, in a sticky set of TGDs, there is no TGD where a marked variable occurs in its body more than once. Thus, every sticky set of TGDs is trivially weakly-sticky.

Example 4. Let $\mathcal{R}$ be the relational schema given in Example 1. Consider the set Σ of TGDs over $\mathcal{R}$ constituted by the following TGDs:

$$\begin{aligned} dept(V,W) &\rightarrow \exists X \exists Y\, emp(W,V,X,Y),\\ emp(V,W,X,Y) &\rightarrow \exists Z\, dept(W,Z), runs(W,Y),\\ runs(W,X), dept(W,Y) &\rightarrow project\_mgr(Y,X). \end{aligned}$$

It is an easy task to verify that $\Pi_F = \mathcal{R} \setminus \{dept[2], emp[4]\}$. Clearly, the variable W that occurs in the body of the third TGD at positions $runs[1]$ and $dept[1]$ is marked. Since both positions are in Π_F, we get that Σ is weakly-sticky. However, due to the existence of W, Σ is not sticky. ■

We continue to study the combined complexity of BCQ answering under weakly-sticky sets of TGDs.

[2] Marked variables are defined as in the Definition 3.

Theorem 3. *BCQ answering under weakly-sticky sets of TGDs is* 2EXPTIME-*complete in combined complexity.*

Proof (sketch). 2EXPTIME-hardness descends immediately from Theorem 1, being weakly-acyclic sets of TGDs a special case of weakly-sticky sets of TGDs. To establish membership in 2EXPTIME in combined complexity, we obtain a 2EXPTIME algorithm for BCQ answering by extending the alternating algorithm Sticky-QAns for sticky sets of TGDs, discussed in Subsection 2.3. The extended algorithm, except the nulls at which the variables in the body of the given query are mapped onto (during the evaluation of the query over the chase), needs also to take into account the nulls appearing at positions in Π_F, i.e., positions with finite rank in the underlying dependency graph. Let N_F be the set of nulls that can appear at positions in Π_F during the chase. Recall that in N_F we can have double-exponentially many nulls. However, each one of these nulls can be represented, using Skolem functions, in exponential space. During the execution of the extended algorithm we need to guess, for each null $z \in N_F$, a Skolem term that represents z. For each null z' at which a variable in the body of the given query is mapped onto, we guess an atom $\underline{a}_{z'}$ which represents the atom in which z' is invented during the chase, as in the algorithm Sticky-QAns, except that $\underline{a}_{z'}$ may contain also Skolem terms that represent nulls of N_F. At each step of the computation of the extended algorithm, we need to keep in memory polynomially many atoms, where each one of these atoms is exponentially large, and thus we need exponential space. This implies that the extended algorithm runs in alternating EXPSPACE, which coincides with 2EXPTIME. □

We conclude this section by establishing PTIME-completeness of BCQ answering under weakly-sticky sets of TGDs in data complexity.

Theorem 4. *Boolean CQ answering under weakly-sticky sets of TGDs is* PTIME-*complete in data complexity.*

Proof (sketch). PTIME-hardness of data complexity immediately follows from PTIME-hardness of fact inference in Datalog programs. Membership in PTIME is established by observing that the nulls appearing at positions in Π_F can be represented, using Skolem functions, in logarithmic space, and thus the extended algorithm discussed in the proof of Theorem 3 runs in alternating LOGSPACE, which coincides with PTIME. □

5 Other Generalizations

Sticky sets of TGDs are arguably a very relevant and applicable modeling tool. Several convincing arguments for the usefulness of sticky sets of TGDs are given in [8]. However, they are not expressive enough for being able to model simple cases such as the TGD $r(X, Y, X) \rightarrow \exists Z\, s(Y, Z)$; clearly, the variable X is marked, and thus the stickiness condition is violated. Note that the above rule falls in the FO-rewritable class of linear TGDs [7]. The question that comes up is whether we can extend sticky sets of TGDs in order to capture also linear TGDs, without losing the desirable property of FO-rewritability. At the first glance it may seen that it could be sufficient to allow a marked variable V to occur more than once in the body of a TGD, as long as V appears only in one body-atom. The obtained class, which for the moment we call *atom-sticky* sets of TGDs, captures linear TGDs. However, as shown by the following example, FO-rewritability is not preserved.

Example 5. Consider the TGD $\sigma = r(X,Y), r(Y,Z) \rightarrow r(X,Z)$. Observe that this rule captures the transitive closure of the relation r, which in general cannot be done using a finite number of first-order queries. Thus, σ is not FO-rewritable. Now, we transform σ into the following set Σ of TGDs:

$$r(X,Y), r(Y',Z) \rightarrow s(X,Y,Y',Z),$$
$$s(X,Y,Y,Z) \rightarrow r(X,Z).$$

Clearly, Σ is not sticky since in the body of the second rule the marked variable Y occurs more than once. However, Y occurs in one atom only, and thus Σ is atom-sticky. Notice that, given a database D, $r(\mathit{chase}(D,\{\sigma\}))$ and $r(\mathit{chase}(D,\Sigma))$ coincide. This implies that if atom-sticky sets of TGDs are FO-rewritable, then transitivity is also FO-rewritable which is a contradiction. ■

It turns out that the class of atom-sticky sets of TGDs proposed above is not only non-FO-rewritable, but is actually undecidable. This can be shown by employing the same principle as in Example 5 to transform an arbitrary set of TGDs into an atom-sticky set. Notice that such transformation was used in [8] to show that the class of *joinless* TGDs, i.e., TGDs where every variable in the body occurs in at most one atom, which is a special case of the proposed class, is undecidable. In the following, we propose a condition somewhat more restrictive than atom-stickiness, which guarantees decidability as well as FO-rewritability.

5.1 Sticky-Join Sets of TGDs

We now introduce the class of *sticky-join* sets of TGDs. Similarly to sticky sets of TGDs, sticky-join sets are defined by a testable condition based on variable-marking. However, the variable-marking for this new class is more sophisticated than the one used for sticky sets. First, we give some auxiliary notions.

Definition 5. *Consider a pair of TGDs σ and σ'. We say that σ is* applicable *to σ' iff the following conditions are satisfied: (i) there exist atoms $\underline{a} \in \mathit{head}(\sigma)$ and $\underline{b} \in \mathit{body}(\sigma')$ such that $\underline{a}$ and $\underline{b}$ unify, (ii) if the term at position π in $\underline{b}$ is a constant, then at position π in $\underline{a}$ we have a $\forall$-variable, and (iii) if at positions $\pi_1, \ldots, \pi_m$, for $m \geqslant 2$, in $\underline{b}$ the same variable occurs, then at positions $\pi_1, \ldots, \pi_m$ in $\underline{a}$ we have either (possibly different) $\forall$-variables, or the same $\exists$-variable.*

The procedure TGD-Expansion accepts as input a set of TGDs Σ, and returns as output a set of TGDs $\Sigma^\star$. TGD-Expansion(Σ) works as follows. Initially, for each $\sigma \in \Sigma$, the TGD σ labeled by $\varnothing$ is added to $\Sigma^\star$. Now, the following step is applied exhaustively (i.e., until a fixpoint is reached): for each pair of TGDs $\langle \sigma, \sigma' \rangle \in \Sigma \times \Sigma^\star$ (including the case where $\sigma = \sigma'$), if σ is applicable to σ' due to the atoms $\underline{a} \in \mathit{head}(\sigma)$ and $\underline{b} \in \mathit{body}(\sigma')$, then let σ^+ be the TGD $\theta(\mathit{body}(\sigma)) \rightarrow \theta(\underline{a})$, where θ is the MGU for $\underline{a}$ and $\underline{b}$. If $\Sigma^\star$ already contains a labeled TGD σ'' isomorphic to σ^+, then the pair $\langle \sigma', \underline{b} \rangle$ is added to the label set of σ''; otherwise, the TGD σ^+ labeled by $\{\langle \sigma', \underline{b} \rangle\}$ is added to $\Sigma^\star$. Each time the above step is applied by considering a pair of TGDs $\langle \sigma, \sigma' \rangle$, to avoid undesirable clutter between variables, the two TGDs σ and σ' are assumed to be standardized apart.

Definition 6. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. The set of TGDs* TGD-Expansion(Σ) *over $\mathcal{R}$ is called the* expanded set *of Σ.*

We now define the procedure SJ-Marking which has as input a set of TGDs Σ, and marks the variables that occur in the body of the TGDs of the expanded set of Σ. SJ-Marking(Σ) works as follows. Let $\Sigma^\star$ be the expanded set of Σ. For each $\sigma \in \Sigma^\star$ and for each variable V in $\mathit{body}(\sigma)$, if there exists an atom $\underline{a}$ in $\mathit{head}(\sigma)$ such that V does not appear in $\underline{a}$, then each occurrence of V in $\mathit{body}(\sigma)$ is marked. Now, the following step is applied exhaustively (i.e., until a fixpoint is reached): for each TGD $\sigma \in \Sigma^\star$ and for each pair $\langle \sigma', \underline{a} \rangle$ in the label set of σ, if a $\forall$-variable V occurs in $\mathit{head}(\sigma)$ at positions $\pi_1, \ldots, \pi_m$, for $m \geqslant 1$, and at each position $\pi_1, \ldots, \pi_m$ in $\underline{a} \in \mathit{body}(\sigma')$ a marked variable occurs, then each occurrence of V in $\mathit{body}(\sigma)$ is marked.

We are now ready, by utilizing the procedure SJ-Marking, to give the formal definition of sticky-join sets of TGDs.

Definition 7. *Consider a set Σ of TGDs over a schema $\mathcal{R}$, and suppose that the expanded set $\Sigma^\star$ of Σ is marked according to* SJ-Marking. *We say that Σ is* sticky-join *iff for each TGD $\sigma \in \Sigma^\star$, there is no marked variable in body(σ) that is in a join, i.e., occurs in (at least) two different atoms.*

Given a sticky set of TGDs Σ (see Definition 3), it can be shown, by induction on the number of the added TGDs to the expansion set $\Sigma^\star$, that each marked variable in the body of a TGD $\sigma \in \Sigma^\star$ occurs just once. This implies that Σ is trivially sticky-join, and thus sticky-join sets of TGDs capture sticky sets. Moreover, given a set Σ of linear TGDs, it is straightforward to see, by construction, that $\Sigma^\star$ is also a set of linear TGDs. Thus, the sticky-join condition is satisfied trivially, which implies that Σ is sticky-join. Note that the sticky-join condition, just like stickiness, ensures that the TGDs are a finite unification set [2].

Example 6. Consider the following set Σ of TGDs:

$$\begin{aligned}
\sigma_1 &: r(X_1, Y_1), p(Z_1, W_1) \;\rightarrow\; s(X_1, Y_1, Z_1, W_1),\\
\sigma_2 &: s(X_2, Y_2, Z_2, Z_2) \;\rightarrow\; \exists W_2\, r(W_2, Y_2), r(X_2, W_2).
\end{aligned}$$

The expanded set $\Sigma^\star$ is as follows:

$$\begin{array}{lll}
\sigma_1 : r(X_1,Y_1),p(Z_1,W_1) \rightarrow s(X_1,Y_1,Z_1,W_1) & \varnothing, \\
\sigma_2 : s(X_2,Y_2,Z_2,Z_2) \rightarrow \exists W_2\, r(W_2,Y_2),r(X_2,W_2) & \varnothing, \\
\sigma_3 : s(X_3,Y_3,Z_3,Z_3) \rightarrow \exists W_3\, r(W_3,Y_3) & \{\langle\sigma_1,1\rangle,\langle\sigma_5,1\rangle\}, \\
\sigma_4 : s(X_4,Y_4,Z_4,Z_4) \rightarrow \exists W_4\, r(X_4,W_4) & \{\langle\sigma_1,1\rangle,\langle\sigma_5,1\rangle\}, \\
\sigma_5 : r(X_5,Y_5),p(Z_5,Z_5) \rightarrow s(X_5,Y_5,Z_5,Z_5) & \{\langle\sigma_2,1\rangle,\langle\sigma_3,1\rangle,\langle\sigma_4,1\rangle\}.
\end{array}$$

It is straightforward to verify that for each $\sigma \in \Sigma^\star$, there is no marked variable in $body(\sigma)$ that occurs in more than one atoms. Therefore, Σ is sticky-join. Observe that $\Pi_\infty = \{r[1], r[2], s[1], s[2]\}$, and hence Σ is not weakly-acyclic. Also, due to the existence of variable Z_2 in $body(\sigma_2)$, Σ is not sticky. Finally, since both occurrences of Z_2 are at positions of Π_∞, Σ is not weakly-sticky. ■

Interestingly, the algorithm Sticky-QAns, proposed for BCQ answering under sticky sets of TGDs (see Subsection 2.3), can be also used for query answering under sticky-join sets of TGDs. The crucial fact is that any new symbols (except the variables in the body of the given query) that may occur in a resolution proof-tree, due to the sticky-join condition, can appear only on a single branch. This observation allows us to establish soundness and completeness of the algorithm, in the case of sticky-join sets of TGDs, and also get the desired upper bounds for data and combined complexity. The lower bound for combined complexity follows immediately from the EXPTIME-hardness of BCQ answering under sticky sets of TGDs. Our main result regarding sticky-join sets of TGDs follows.

Theorem 5. *Sticky-join sets of TGDs are FO-rewritable. Also, BCQ answering under sticky-join sets of TGDs is* EXPTIME-*complete in combined complexity.*

Analogously to weakly-sticky sets of TGDs, we can define the class of *weakly-sticky-join* sets of TGDs, that generalizes both weakly-acyclic and sticky-joins sets of TGDs. The same algorithm employed for BCQ answering under weakly-sticky sets of TGDs, can be also used for query answering under weakly-sticky-join sets. The next result follows immediately.

Theorem 6. *BCQ answering under weakly-sticky-join sets of TGDs is* PTIME-*complete and* 2EXPTIME-*complete in data and combined complexity, respectively.*

It is not difficult to show that we can incorporate non-recursive Datalog rules in our framework, without affecting data and combined complexity, since they can be "chased" separately. In particular, on the top of the non-recursive Datalog program, we can apply our techniques for (weakly-)sticky-join sets of TGDs.

Acknowledgments. This work was supported by the European Research Council under the European Community's 7-th Framework Programme (FP7/2007-2013)/ERC grant no. 246858 DIADEM. The authors also acknowledge support by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). Georg Gottlob acknowledges a Royal Society Wolfson Research Merit Award.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading
2. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: Extending decidable cases for rules with existential variables. In: Proc. of IJCAI, pp. 677–682 (2009)
3. Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: Proc. of ICALP, pp. 73–85 (1981)
4. Cabibbo, L.: The expressive power of stratified logic programs with value invention. Inf. Comput. 147(1), 22–56 (1998)
5. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: Proc. of KR, pp. 70–80 (2008)
6. Calì, A., Gottlob, G., Lukasiewicz, T.: Datalog$^\pm$ A unified approach to ontologies and integrity constraints. In: Proc. of ICDT, pp. 14–30 (2009)
7. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. In: Proc. of PODS, pp. 77–86 (2009)
8. Calì, A., Gottlob, G., Pieris, A.: Advanced processing for ontological queries. In: Proc. of VLDB (to appear, 2010)
9. Calì, A., Gottlob, G., Pieris, A.: Advanced processing for ontological queries (2010), `http://benner.dbai.tuwien.ac.at/staff/gottlob/CGP.pdf`
10. Calì, A., Gottlob, G., Pieris, A.: Ontological reasoning with F-Logic Lite and its extensions. In: Proc. of AAAI (to appear, 2010)
11. Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proc. of PODS, pp. 260–271 (2003)
12. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-lite family. J. Autom. Reasoning 39(3), 385–429 (2007)
13. Dantsin, E., Eiter, T., Georg, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. 33(3), 374–425 (2001)
14. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisisted. In: Proc. of PODS, pp. 149–158 (2008)
15. Deutsch, A., Tannen, V.: Reformulation of XML queries and constraints. In: Proc. of ICDT, pp. 225–241 (2003)
16. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
17. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. 28(1), 167–189 (1984)
18. Kolaitis, P.G., Panttaja, J.: Personal Communication (2009)
19. Kolaitis, P.G., Panttaja, J., Tan, W.-C.: The complexity of data exchange. In: Proc. of PODS, pp. 30–39 (2006)
20. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. 4(4), 455–469 (1979)
21. Mailharrow, D.: A classification and constraint-based framework for configuration. Artif. Intell. for Engineering Design, Analysis and Manufacturing 12(4), 383–397 (1998)
22. Panttaya, J.: Complexity in databases, games, and logics. PhD thesis, University of California Santa Cruz (2006)
23. Patel-Schneider, P.F., Horrocks, I.: A comparison of two modelling paradigms in the semantic web. J. Web Semantics 5(4), 240–250 (2007)
24. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semantics 10, 133–173 (2008)

Data Validation with OWL Integrity Constraints*

(Extended Abstract)

Evren Sirin

Clark & Parsia, LLC, Washington, DC, USA
evren@clarkparsia.com

Abstract. Data validation is an important part of data integration and analysis tasks. The consequences of having invalid data ranges from rather harmless application failures to serious errors in decision making process. Web Ontology Language (OWL) provides an expressive language that facilitates data integration and analysis tasks. However, the Open World Assumption (OWA) adopted by standard OWL semantics, combined with the absence of the Unique Name Assumption (UNA), makes it difficult to use OWL for data validation. What triggers constraint violations in closed world systems leads to new inferences in standard OWL systems. In this paper, we present an Integrity Constraint (IC) semantics for OWL axioms to address this issue. Ontology modelers can choose which axioms will be interpreted with IC semantics and combine open world reasoning with closed world constraint validation in a flexible way. We also show that IC validation can be reduced to query answering under certain conditions.

1 Introduction

Data integration and analysis are important tasks in many domains and applications. As IT systems are moving towards a more distributed pattern of implementation and deployment, applications need data to be enriched with more semantics. Richer data semantics enables us to build a unified model over distributed data sources and to perform analysis and reasoning tasks over the unified model. Web Ontology Language (OWL) provides a solution to this problem by allowing the representation of data semantics in a formal logic-based language that is amenable to automated reasoning.

The semantics of OWL addresses distributed knowledge representation scenarios where complete knowledge about the domain cannot be assumed. OWL adopts Open World Assumption (OWA) so a statement cannot be inferred to be false on the basis of failure to prove it. Furthermore, OWL does not adopt Unique Name Assumption (UNA) which means two resources with different identifiers might be treated as same objects.

* This paper is a summary of earlier publications [8,9].

The above characteristics of OWL make it difficult to use OWL for data validation in applications where complete knowledge can be assumed for some or all parts of the domain. In such data-centric applications, we would like to use OWL as an expressive schema language to specify the constraints that must be satisfied by instance data.

In the literature, OWA has been identified as "the biggest single hurdle to understanding OWL" [7]. It is a common misconception for newcomers to think that axioms in OWL are similar to constraints in relational databases. However, the axioms in an ontology are meant to *infer new knowledge* rather than *trigger an inconsistency*.

In many use cases, we need the ability to combine open world reasoning with closed world constraint validation in a flexible way. It should be possible to use OWA for the parts of the domain where complete knowledge cannot be assumed and use CWA for the other parts of the domain where we have complete knowledge.

In our previous work [9], we presented an alternative semantics for OWL axioms to enable closed world data validation. The ontology developers can choose which axioms will be interpreted with regular OWL semantics and which axioms will be interpreted with IC semantics.

In the rest of this paper, we provide a simple example to illustrate the difficulties in using OWL for data validation, briefly describe our IC semantics proposal and present an IC validation algorithm we developed for this semantics. Most of the technical details are omitted in this paper and can be found in [9].

2 IC Example

There are various types of ICs identified in the literature. Some examples are subsumption constraints, typing constraints, participation constraints and uniqueness constraints. We will use participation constraints as one example to illustrate the difficulties in using OWL for data validation.

A mandatory participation constraint states that instances of the constrained class should participate in a relation. If we would like to express that every `Product` instance should be related by the `madeBy` property to a there is a `Manufacturer`, we can write the following OWL axiom:

$$\texttt{Product} \sqsubseteq \exists \texttt{madeBy}.\texttt{Manufacturer} \tag{1}$$

However, this participation constraint is expressing a general truth about the world. It does not constrain what should exist in a specific ontology or knowledge base. For example, suppose we have an ontology where there is an instance of the `Product` class is defined:

$$\texttt{Product}(\texttt{product1}) \tag{2}$$

This ontology is not inconsistent according to the semantics of OWL since with OWA we can conclude that `product2` has a manufacturer but no knowledge about that manufacturer exists in this ontology. For data validation purposes, we might want to detect or even prevent cases when the manufacturer of a product is not known. However, it is not possible to do so with the above axiom.

It is possible to "close" the onotlogy by augmenting it with additional assertions to state that all the relevant information is known. In a preprocessing step, we can check the explicit and the implicit property values for each individual and add explicit cardinality restrictions to assert that there are no more property values. For the above example, we can add the following type assertion:

$$(\leq 0 \ \texttt{madeBy})(\texttt{product1}) \tag{3}$$

The combination of (1), (2), and (3) would result in an inconsistency with regular OWL semantics.

However, there are couple of problems with the preprocessing approach. First, the preprocessing step can be computationally expensive especially because we need to take the entailments of a KB into consideration. This problem becomes more significant if the data assertions are changing frequently. Second, even if we ignore the efficiency considerations, preprocessing solution cannot address other kind of constraints such as typing constraints (see examples discussed in [9]).

3 IC Semantics for OWL

It is apparent, even from the simple example presented above, that OWL ontologies cannot be used in a straight-forward way for data validation purposes. In order to overcome this problem, we started investigating possible solutions. Our goal was to enable using OWL to express ICs without needing a different representation language.

The approach formalized in [4] describes one such solution where the axioms in an OWL ontology is partitioned into two sets. The axioms in one set is interpreted with regular OWL axioms to do inference and the axioms in the second set is interpreted with a closed-world semantics based on minimal Herbrand models to do validation. Even tough this approach satisfies many of the conditions mentioned so far we have identified several issues with the semantics that would yield undesirable results in practice [9].

In order to overcome the problems we have identified in existing solutions, we defined a new IC semantics [9] for interpreting OWL axioms. The IC semantics we define extends the model theory of OWL 2 or more correctly the model theory of the Description Logic $\mathcal{SROIQ}$ [2] which is the basis of OWL 2 semantics [5].

The semantics extension we propose has many similarities to epistemic DLs such as $\mathcal{ALCK}$[1] but there are also some differences. First, unlike $\mathcal{ALCK}$ we do not require the epistemic operator **K** to be explicitly used. The semantics for ICs is defined as if the **K** operator exists in front of every class and property. Second, the IC semantics we define is applicable to any SROIQ ontology and not restricted to $\mathcal{ALC}$ expressivity.

Third and most importantly, $\mathcal{ALCK}$ semantics adopt strict UNA which excludes the possibility of stating two names identify the same individual. Even though we want to prohibit ICs to infer equality between individuals, we still would like to allow OWL ontologies to include explicit equality between individual names to assist data integration scenarios. In our formalization, we adopt a weak form of UNA where two different individual names are assumed to be different unless their equality is required to satisfy the axioms in an ontology.

4 Data Validation Algorithm

We show in [9] that given a regular OWL ontology and an IC expressed as an OWL axiom, checking if the IC is violated by the OWL ontology can be reduced to conjunctive query answering under certain conditions. These conditions require, intuitively, that either the ontology does not contain disjunctive (in)equality between individuals or the IC does include cardinality restrictions. In OWL, only nominals or cardinality restrictions can result in disjunctive (in)equality so if neither exists in an ontology then we can use the query reduction technique as explained next.

The ICs can be translated to queries using an approach very similar to the well-known Lloyd-Topor transformation [3].The queries that are produced as the result of this transformation contains only distinguished variables and may also include the negation-as-failure operator. The translation algorithm transforms an IC into a query such that the IC is violated w.r.t. the proposed IC semantics by an ontology if and only if the ontology entails the query. In other words, whenever the answer set of the query is not empty w.r.t. an ontology, we conclude that the IC is violated by that ontology.

There are some nice practical benefits of the query translation approach. It is possible to express the generated queries using the SPARQL [6] query language which is the most commonly used Semantic Web query language. Even tough the standard SPARQL semantics is not compatible with OWL semantics, it allows extended entailment regimes to be used and a precise definition of OWL-compatible SPARQL semantics is being developed by the W3C's SPARQL Working Group as part of SPARQL 1.1.[1] Most existing OWL reasoners support answering SPARQL queries so the SPARQL queries generated by this translation can be evaluated using off-the-shelf OWL reasoners.

5 Conclusions

The IC semantics we propose addresses many IC use cases like such as participation, typing and uniqueness constraints. By adopting weak form of UNA, the IC semantics allows explicit equality assertions to be asserted while avoiding undesirable equality inferences due to uniqueness constraints. We have shown in [9] with several examples that our IC semantics proposal provides more intuitive results compared to other proposals such as [4]. Our approach allows ontology developers use OWL both to express axioms for inferencing and ICs for data validation

[1] http://www.w3.org/TR/sparql11-entailment/

providing a flexible way to combine open world reasoning with closed world data validation. Validation of ICs can be reduced to query answering where a straightforward translation algorithm transforms ICs to SPARQL queries. This translation allows us to use existing OWL reasoners for IC validation.

Acknowledgements

This paper summarizes work done in collaboration with Jiao Tao and published in [8,9]. This work was partly funded by the NIST grant SB1342-08-SE065. I would also like to acknowledge valuable feedback and input from Héctor Pérez-Urbina, Peter Patel-Schneider, Evan Wallace, and Kendall Clark.

References

1. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: An epistemic operator for description logics. AI 100(1-2), 225–274 (1998)
2. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible sroiq. In: KR 2006, pp. 57–67. AAAI Press, Menlo Park (2006)
3. Lloyd, J.W., Topor, R.W.: Making prolog more expressive. J. of Logic Programming 1, 225–240 (1984)
4. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between owl and relational databases. In: WWW 2007, pp. 807–816. ACM Press, New York (2007)
5. Motik, B., Patel-Schneider, P.F., Grau, B.C.: Owl 2 web ontology language direct semantics (2009), http://www.w3.org/TR/owl2-semantics/
6. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF (2008), http://www.w3.org/TR/rdf-sparql-query/
7. Rector, A.L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) EKAW 2004. LNCS (LNAI), vol. 3257, pp. 63–81. Springer, Heidelberg (2004)
8. Sirin, E., Tao, J.: Towards integrity constraints in owl. In: Proceedings of the Workshop on OWL: Experiences and Directions, OWLED 2009 (2009)
9. Tao, J., Evren, S., Bao, J., McGuiness, D.: Integrity constraints in owl. In: 24th Conference on Artificial Intelligence, AAAI 2010 (2010)

SPARQL1.1: New Features and Friends (OWL2, RIF)

Axel Polleres[*]

Digital Enterprise Research Institute, National University of Ireland, Galway
axel.polleres@deri.org

Abstract. In this tutorial we will give an overview of new features in SPARQL 1.1, which the W3C is currently working on, as well as on the interplay with its ”neighbour standards”, OWL2 and RIF. We will also give a rough overview of existing implementations to play around with.

1 Research on and Experiences with SPARQL1.0

The availability of a standard query language for RDF, namely SPARQL [22], which is a W3C standard recommendation since early 2008, has proven to be a crucial factor for the recent wide uptake of Semantic Web technologies at large. In informal terms, SPARQL plays the same role for the Semantic Web as SQL does for relational data. SPARQL's syntax is roughly inspired by Turtle [3] and SQL [26], providing basic means to query RDF, such as unions of conjunctive queries, value filtering, optional query parts, as well as slicing and sorting results.

The formal semantics of SPARQL is very much inspired by academic results, such as by the seminal papers of Pérez et al. [17,19]. Their work further lead to refined results on equivalences within SPARQL [23] and on the relation of SPARQL to Datalog [20]. Angles and Gutierrez [2] later showed that SPARQL has exactly the expressive power of non-recursive safe Datalog with negation. However, there are also subtle differences between these theoretical works and the semantics as defined in the official W3C specification, such as the treatment of FILTER expressions in OPTIONAL, discussed in [2] or SPARQL's multi-set semantics, as opposed to the set-based algebra presented in [17].

Now, two years after recommendation as a standard by W3C, a wide range of vendors and implementations support SPARQL in its original specification.[1]

2 New Features to Come in SPARQL1.1

Many of the existing SPARQL implementations and various proposals in the academic literature include extensions, to fit customer needs or specific use cases, such as aggregates, sub-queries, negation, path expressions, and many more. The recently re-chartered SPARQL W3C working group[2] has picked several of these features to be

[*] The author's work is funded by the Science Foundation Ireland (SFI) project Lion-2 (SFI/08/CE/I1380). We particularly thank all the members of the SPARQL1.1 working group.

[1] The W3C wiki at http://esw.w3.org/SparqlImplementations gives a good entry point to existing SPARQL implementations.

[2] http://www.w3.org/2009/sparql/wiki

included in the next version of the standard, SPARQL1.1, listed in the working group charter.[3]

The new features we will focus on in this tutorial are:

Aggregate functions. Aggregate functions will allow operations on the query engine side such as counting, numerical min/max/average and so on, by operating over columns of results. This feature is commonly known from other query languages such as SQL, but also well investigated in terms of extensions of Datalog, cf. for instance [8]. A proposal to extend SPARQL with aggregates following these ideas for Datalog has been made in [21], whereas most major implementations and also the current design discussed in the SPARQL1.1 working group rather follow the SQL design.

Subqueries. This feature will allow nesting the results of a query within another query.

Negation. In the current SPARQL Recommendation "Negation As Failure" is possible by a non-intuitive combination of OPTIONAL patterns and FILTERs. In SPARQL1.1 a dedicated language construct for expressing negation shall be introduced.

Project expressions. This feature will allow one to return the values of expressions over result bindings, rather than just RDF terms bound in the queried graph. Most common implementations support this feature in one or the other way.

Property paths. Many classes of queries over RDF graphs require searching hierarchical data structures and involve arbitrary-length paths through the graphs. Examples include retrieving all the elements of an RDF collection, searching for the direct and indirect superclasses of a class, etc. It is not possible to express such queries using the original SPARQL recommendation. Again, different implementations provide different mechanisms to enable such path expressions in one form or the other, as well as some respective proposals have been made in the literature [1,18].

Entailment regimes. SPARQL1.0 has in only been defined as a query language over RDF graphs, not taking into account RDF Schema, OWL ontologies, or custom rule-based inference regimes, as implemented by existing engines. Although the original specification defines frame conditions for extending SPARQL by higher entailment regimes [22, Section 12.6], few works (e.g. [13]) have actually instantiated this mechanism and defined how SPARQL should handle ontologies and rule sets. SPARQL1.1 aims to close this gap. Note that this issue is closely related to ongoing research on conjunctive query answering over expressive description logics in the Description Logics community [7,10,9,15], none of which yet having covered the Description Logics underlying OWL and OWL2, i.e. $\mathcal{SHOIN}$(D) and $\mathcal{SROIQ}$(D). Answering full SPARQL queries on top of OWL has only preliminarily been addressed in the scientific community [25,14] so far. As we will see, SPARQL1.1 thus needs to make some pragmatic choices – essentially, disallowing non-distinguished variables.

In the tutorial we will exemplify these features and discuss the current design choices within the SPARQL1.1 working group, particularly the relationship of SPARQL1.1 to

[3] `http://www.w3.org/2009/05/sparql-phase-II-charter.html`

its very recent "neighbour standards" in the Semantic Web architecture, the Rule Interchange Format (RIF) [4,5,6] and the latest version of the Web Ontology Language (OWL2) [12,16]. We will also try to sketch how some of the theoretical results for SPARQL1.0 discussed in the previous Section may carry over to SPARQL1.1. The relevant current working drafts of the SPARQL working group include:

– SPARQL 1.1 Query Language [24],
– SPARQL 1.1 Entailment Regimes [11]

References

1. Alkhateeb, F., Baget, J.-F., Euzenat, J.: Extending sparql with regular expression patterns (for querying rdf). Journal of Web Semantics 7(2), 57–73 (2009)
2. Angles, R., Gutierrez, C.: The expressive power of sparql. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 114–129. Springer, Heidelberg (2008)
3. Beckett, D., Berners-Lee, T.: Turtle – Terse RDF Triple Language. W3c team submission, W3C (January 2008), `http://www.w3.org/TeamSubmission/turtle/`
4. Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A., Reynolds, D.: RIF Core Dialect. W3C recommendation, W3C (June 2010), `http://www.w3.org/TR/rif-core/`
5. Boley, H., Kifer, M.: RIF Basic Logic Dialect. W3C recommendation, W3C (June 2010), `http://www.w3.org/TR/rif-bld/`
6. de Bruijn, J.: RIF RDF and OWL Compatibility. W3C recommendation, W3C (June 2010), `http://www.w3.org/TR/rif-rdf-owl/`
7. Eiter, T., Lutz, C., Ortiz, M., Simkus, M.: Query answering in description logics with transitive roles. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California, USA, pp. 759–764 (July 2009)
8. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
9. Glimm, B., Horrocks, I., Sattler, U.: Unions of conjunctive queries in shoq. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, pp. 252–262. AAAI Press, Menlo Park (September 2008)
10. Glimm, B., Lutz, C., Horrocks, I., Sattler, U.: Conjunctive query answering for the description logic shiq. J. Artif. Intell. Res (JAIR) 31, 157–204 (2008)
11. Glimm, B., Ogbuji, C., Hawke, S., Herman, I., Parsia, B., Polleres, A., Seaborne, A.: SPARQL 1.1 Entailment Regimes. W3C working draft, W3C (June 2010), `http://www.w3.org/TR/sparql11-entailment/`
12. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: OWL 2 Web Ontology Language Primer. W3c recommendation, W3C (October 2009), `http://www.w3.org/TR/owl2-primer/`
13. Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: Dynamic querying of mass-storage rdf data with rule-based entailment regimes. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 310–327. Springer, Heidelberg (2009)
14. Jing, Y., Jeong, D., Baik, D.-K.: SPARQL graph pattern rewriting for OWL-DL inference queries. Knowl. Inf. Syst. 20(2), 243–262 (2009)

15. Krötzsch, M., Rudolph, S., Hitzler, P.: Conjunctive queries for a tractable fragment of owl 1.1. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 310–323. Springer, Heidelberg (2007)
16. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C., Calvanese, D., Carroll, J., Giacomo, G.D., Hendler, J., Herman, I., Parsia, B., Patel-Schneider, P.F., Ruttenberg, A., Sattler, U., Schneider, M.: OWL 2 Web Ontology Language Profiles. W3c recommendation, W3C (October 2009), `http://www.w3.org/TR/owl2-profiles/`
17. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 30–43. Springer, Heidelberg (2006)
18. Pérez, J., Arenas, M., Gutierrez, C.: nsparql: A navigational language for rdf. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 66–81. Springer, Heidelberg (2008)
19. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Transactions on Database Systems 34(3), 45 pages, Article 16 (2009)
20. Polleres, A.: From SPARQL to rules (and back). In: Proceedings of the 16th World Wide Web Conference (WWW 2007), Banff, Canada, pp. 787–796. ACM Press, New York (May 2007)
21. Polleres, A., Scharffe, F., Schindlauer, R.: SPARQL++ for mapping between RDF vocabularies. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 878–896. Springer, Heidelberg (2007)
22. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3c recommendation, W3C (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`
23. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: 13th International Conference on Database Theory (ICDT 2010), Lausanne, Switzerland (March 2010)
24. Seaborne, A., Harris, S.: SPARQL 1.1 Query Language. W3C working draft, W3C (June 2010), `http://www.w3.org/TR/sparql11-query/`
25. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria. CEUR-WS.org (June 2007)
26. SQL-99. Information Technology - Database Language SQL- Part 3: Call Level Interface (SQL/CLI). Technical Report INCITS/ISO/IEC 9075-3, INCITS/ISO/IEC, Standard specification (October 1999)

Usability of a Visual Language for DL Concept Descriptions*

Fernando Náufel do Amaral

LLaRC – Laboratório de Lógica e Representação do Conhecimento,
Depto. de Ciência e Tecnologia, Pólo Universitário de Rio das Ostras,
Universidade Federal Fluminense, Rio das Ostras, RJ, Brazil
fnaufel@ic.uff.br

Abstract. The development and use of ontologies may require users with no training in formal logic to handle complex concept descriptions. To aid such users, we propose a new visualization framework called "model outlines", where more emphasis is placed on the *semantics* of concept descriptions than on their *syntax*. We have conducted a usability study comparing model outlines and Manchester OWL, with results that indicate the potential benefits of our visual language for understanding concept descriptions.

1 Introduction

When working with formal ontologies, one often needs to formally represent conditions for membership in the defined classes. In this paper, we will call such conditions *concept descriptions*, following the description logic (DL) tradition [1].

Concept descriptions are important in many scenarios related to ontology development and use. For example, DL reasoners perform logical inferences by manipulating concept descriptions according to a specific deductive calculus. In many cases, users may be interested not only in the answers provided by such reasoners, but also in the chains of reasoning that led to those answers. In order to understand such chains of reasoning, users must be able to understand the meaning of the concept descriptions involved. This area of study is referred to as *proof explanation* [2].

Another situation where concept descriptions play an important role is in the definition and use of *ontology query languages* [3]; here, building a query may include writing modified concept descriptions that contain free variables (representing individuals that must be returned by the query).

Because many users of formal ontologies have no specific training in logic, the problem of representing concept descriptions in a user-friendly fashion is an important one, and many researchers have proposed different ways of solving it: replacing logical symbols with keywords in DL languages [4], automatically generating natural language paraphrases of concept descriptions [5], or using diagrams [6,7].

* Work partially supported by research grant E-26/112.038/2008 from FAPERJ.

As an example to make this discussion more concrete, consider the following concept description in DL syntax (to be formally introduced in Sect. 2 below), which appears in [8], a paper about proof explanation:

$$\exists hasChild.\top \sqcap \forall hasChild.\neg((\exists hasChild.\neg Doctor) \sqcup (\exists hasChild.Lawyer)) \quad (1)$$

Diagrammatic representations of concept descriptions have given rise to implementations of "visual" ontology browsers. One such example is the visualization tool GrOWL [9], which produces the diagram in Fig. 1 for the concept description in (1). As can be seen, the diagram is essentially an abstract syntax tree, which offers nonspecialist users little help in understanding the semantics of the description, especially if those users are not familiar with the DL symbols "∃", "∀", "¬" and "⊔". In fact, we have found this to be a common phenomenon: many visualization frameworks for concept descriptions are too faithful to the *syntax* of the representation languages (e.g., DL, OWL), a feature which may prevent users from grasping the *semantics* of the concept descriptions.

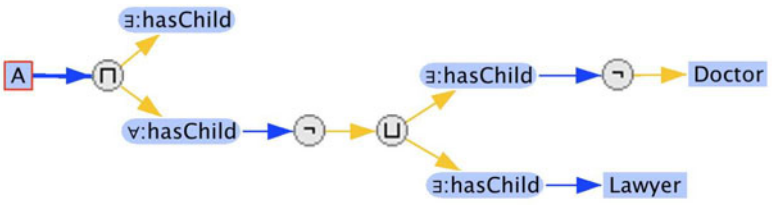


Fig. 1. Diagram produced for (1) by GrOWL [9] (manually laid out)

This paper discusses *model outlines*, which depart from the syntax-based tradition in that they consist of diagrams characterizing the class of *models* of a given concept description. (Here, we use the term "model" in the logical sense.) The model outline for (1), produced after applying a carefully defined set of simplification rules to the original concept description, is presented in Fig. 2. By adhering to some simple graphical conventions, a user can understand that the concept description represents a set of individuals having at least one child and having as grandchildren (if any) only doctors and non-lawyers.

Our previous papers [10,11] introduced the first version of model outlines and compared them to natural language paraphrases of concept descriptions. Since then, we have reformulated the visual language so as to make it more intuitive (e.g., including optional labeled clusters, rendering cardinality restrictions as text and fine-tuning the placement of inner boxes). We have also altered the conversion algorithms to conform to the new visual language.

Most importantly, we have conducted a first usability test of model outlines, with promising results. Users from different backgrounds were shown concept descriptions in two formalisms: our model outlines and Manchester OWL

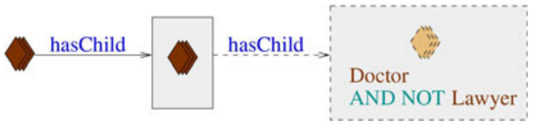


Fig. 2. Model outline for description (1)

(a textual notation for DL which uses keywords for logical symbols, infix notation for restrictions, syntax highlighting and indentation in order to make descriptions more readable for nonspecialists — see [4]). We then tested ease of understanding for each formalism by asking the users questions about the concept descriptions shown.

This paper is structured as follows: Sect. 2 presents the syntax of model outlines for the description logic $\mathcal{ALCN}$, at the concrete (token) and at the abstract (type) levels, as is suitable for diagrammatic systems [12]; Sect. 3 defines the precise semantics of model outlines, in the form of algorithms that translate from model outlines to $\mathcal{ALCN}$ concept descriptions; Sect. 4 discusses the translation of $\mathcal{ALCN}$ concept descriptions to model outlines; Sect. 5 reports and analyzes the results of the usability test; Sect. 6 contains our concluding remarks.

2 Syntax of Model Outlines

We consider the description logic $\mathcal{ALCN}$, whose language of concept descriptions[1] is specified in Fig. 3, both in the DL syntax and in Manchester OWL. There, A stands for a class name (i.e., an atomic concept term), R stands for a property name (i.e., an atomic role term), and n represents a natural number. The (set-theoretical) meaning of these descriptions is given by a nonempty set Δ (the *universe* or *domain*) along with an interpretation $\mathcal{I}$ mapping each concept description C to a set $\mathcal{I}(C) \subseteq \Delta$, and each role term R to a binary relation $\mathcal{I}(R) \subseteq \Delta \times \Delta$. An interpretation $\mathcal{I}$ must map each description in the first two columns to the set in the third column. $\#S$ denotes the cardinality of a set S. A *literal* is a description of the form A or of the form $\neg A$, where A is an atomic concept term.

The *concrete syntax* of model outlines defines their physical representation. What follows is an informal definition: a model outline contains *clusters* (*solid* or *dashed*), *arrows* (*solid* or *dashed*) and *boxes*. The *root* of the model outline is a solid cluster. A cluster may have an optional *class label* below it, consisting of a disjunction or of a conjunction of literals. So may a box. A box may also have an optional *cardinality label* below it, which may be of the form "(from m thru n)", "(m or more)", or "(exactly m)", with m, n natural numbers, $m < n$. The *source* of an arrow may be a cluster or a box. The *target* of an arrow is always a box. Each

[1] Work is under way to define model outlines for more expressive languages, such as the concept language underlying OWL 2 [13].

DL	Manchester	Meaning
$C, D \rightarrow A$	A	$\mathcal{I}(A)$
$\mid \top$	**THING**	Δ
$\mid \bot$	**NOTHING**	$\varnothing$
$\mid \neg C$	**NOT** C	$\Delta - \mathcal{I}(C)$
$\mid C \sqcap D$	C **AND** D	$\mathcal{I}(C) \cap \mathcal{I}(D)$
$\mid C \sqcup D$	C **OR** D	$\mathcal{I}(C) \cup \mathcal{I}(D)$
$\mid \forall R.C$	R **ONLY** C	$\{a \in \Delta \mid \forall b.[(a,b) \in \mathcal{I}(R) \Rightarrow b \in \mathcal{I}(C)]\}$
$\mid \exists R.C$	R **SOME** C	$\{a \in \Delta \mid \exists b.[(a,b) \in \mathcal{I}(R) \wedge b \in \mathcal{I}(C)]\}$
$\mid \leq n.R$	R **MAX** n	$\{a \in \Delta \mid \#\{b \mid (a,b) \in \mathcal{I}(R)\} \leq n\}$
$\mid \geq n.R$	R **MIN** n	$\{a \in \Delta \mid \#\{b \mid (a,b) \in \mathcal{I}(R)\} \geq n\}$
$\mid = n.R$	R **EXACTLY** n	$\{a \in \Delta \mid \#\{b \mid (a,b) \in \mathcal{I}(R)\} = n\}$

Fig. 3. $\mathcal{ALCN}$ concept descriptions and their meaning

box is the target of exactly one arrow. An arrow must have a *role label* above it, consisting of a role name. A box *contains* one or more clusters, according to constraints that we do not include in this informal description, but which will be made explicit in the abstract syntax below. A box may also contain at most one *"among-which" inner box*, which in turn contains one or more clusters, all of them solid. Inner boxes are never the source of arrows. A box or a cluster may have a *case widget* above it.

Fig. 4 shows an example model outline. The target box of the arrow labeled "*hasAttendance*" has both a class label ("*Enrolled*") and a cardinality label ("from 10 to 50"). The target box of the arrow labeled "*hasAttendance*" also has an "among-which" inner box. This model outline does not have case widgets.

At this point, the reader should test the appropriateness of the choice of visual presentation of the components of model outlines. We suggest that the reader (without any further knowledge of the meaning of these components) formulate a natural language description of the constraints imposed upon the individuals of class *GraduateCourse* at the root of the outline. If the reader is knowledgeable in DL syntax, the reader should also produce an $\mathcal{ALCN}$ concept description. In Sect. 3 below, we explain the precise meaning of this model outline, and in Sect. 4 we show the steps involved in its construction.

Case widgets indicate alternatives (i.e., disjunction). If a cluster or a box has a case widget above it, the user may browse the different cases interactively, one case at a time, by clicking on the triangles on either side of the case widget.

In Fig. 5, for example, there are 4 cases altogether, specifying objects that are either (a) *Book*s having all extras (if any) translated to *Portuguese* (and possibly other languages), or (b) *Book*s having all extras (if any) in *Audio* format (and possibly other formats), or (c) *ClassNotes* having at least one *Free* copy in *PDF* format (and possibly other formats, and other copies), or (d) *ClassNotes* having at least one *Low*-priced copy (and possibly other copies).

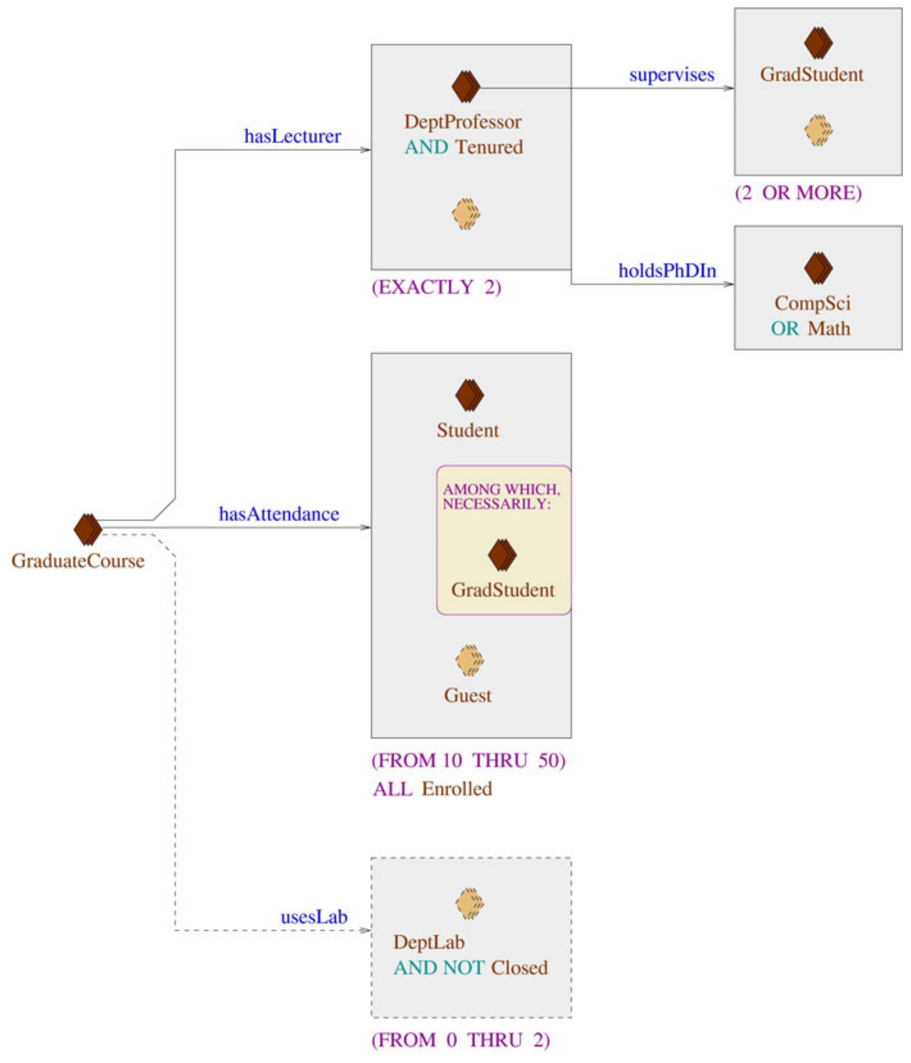


Fig. 4. Example model outline

More formally, the model outline in Fig. 5 corresponds to the description

$$[Book \sqcap \forall hasExtras.(\exists hasTranslation.Portuguese \sqcup \exists hasFormat.Audio)] \sqcup$$
$$\{ClassNotes \sqcap$$
$$\exists hasCopy.[(Free \sqcap \exists hasFormat.PDF) \sqcup (\exists hasPrice.Low \sqcap \forall hasPrice.Low)]\}$$

As for the *abstract syntax*, a model outline is formally defined as a LISP-style list generated by the grammar in Fig. 6, in extended BNF notation. The list

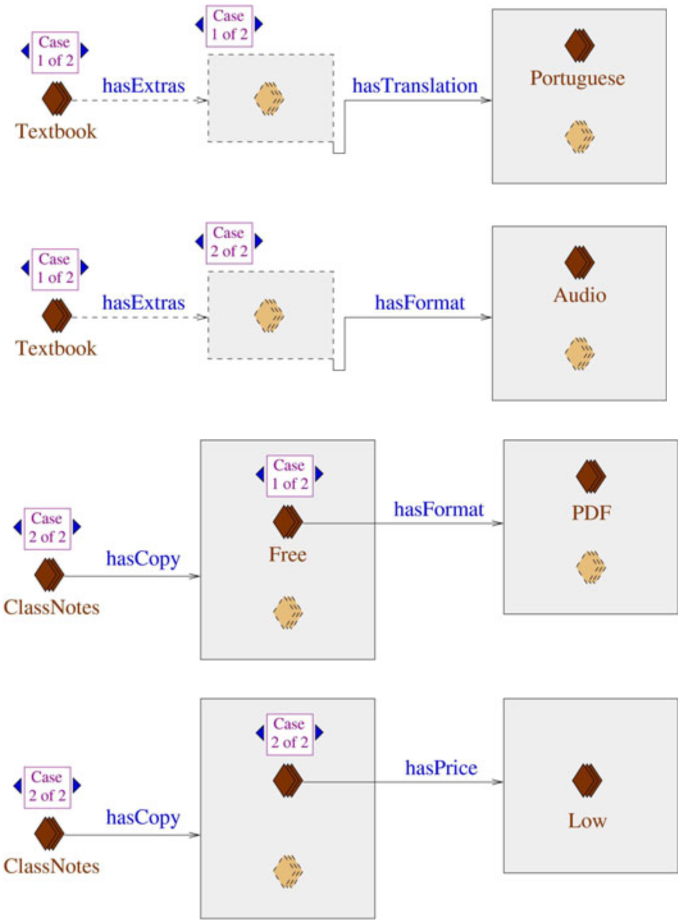


Fig. 5. Example model outline with case widgets

representation is not meant for human consumption, but rather for automatic processing by algorithms such as the ones presented in the next section.

3 Semantics of Model Outlines

The appearance of the components of a model outline follows some (hopefully intuitive) graphical conventions:

Individuals are represented by clusters of diamonds. The presence of a cluster (as opposed to a single diamond) emphasizes the idea that one *or more* individuals may appear in a given situation. E.g., in Fig. 4, the graduate courses in question may have as lecturers more than one tenured department professor holding a CompSci or Math PhD degree and supervising at least one graduate student from a total of 2 or more individuals.

$$
\begin{aligned}
\langle \mathit{outline}\rangle &\rightarrow \langle \mathit{solidClstrCases}\rangle \\
\langle \mathit{solidClstrCases}\rangle &\rightarrow \textbf{(cases } \langle \mathit{solidCluster}\rangle^{+} \textbf{)} \\
\langle \mathit{solidCluster}\rangle &\rightarrow \textbf{(cluster solid } \langle \mathit{classLabel}\rangle \textbf{ (} \langle \mathit{arrow}\rangle^{\star} \textbf{))} \\
\langle \mathit{classLabel}\rangle &\rightarrow \textbf{( )} \mid \textbf{(} \langle \mathit{literal}\rangle \textbf{)} \\
&\mid \textbf{ (and } \langle \mathit{literal}\rangle\ \langle \mathit{literal}\rangle^{+} \textbf{)} \mid \textbf{(or } \langle \mathit{literal}\rangle\ \langle \mathit{literal}\rangle^{+} \textbf{)} \\
\langle \mathit{literal}\rangle &\rightarrow \langle \mathit{conceptName}\rangle \mid \textbf{(not } \langle \mathit{conceptName}\rangle \textbf{)} \\
\langle \mathit{arrow}\rangle &\rightarrow \textbf{(arrow solid } \langle \mathit{roleName}\rangle \textbf{ (} \langle \mathit{intrvl}\rangle^{\star} \textbf{) } \langle \mathit{solidBoxCases}\rangle \textbf{)} \\
&\mid \textbf{ (arrow dashed } \langle \mathit{roleName}\rangle \textbf{ (} \langle \mathit{intrvl}\rangle^{\star} \textbf{) } \langle \mathit{dashedBoxCases}\rangle \textbf{)} \\
\langle \mathit{intrvl}\rangle &\rightarrow \textbf{(} \langle \mathit{number}\rangle\ \langle \mathit{number}\rangle \textbf{)} \mid \textbf{(} \langle \mathit{number}\rangle \textbf{ infty)} \\
\langle \mathit{solidBoxCases}\rangle &\rightarrow \textbf{(cases } \langle \mathit{solidBox}\rangle^{+} \textbf{)} \\
\langle \mathit{solidBox}\rangle &\rightarrow \textbf{(box } \langle \mathit{classLabel}\rangle \textbf{ (} \langle \mathit{solidClstrCases}\rangle^{+} \textbf{) } \langle \mathit{opt}\rangle \textbf{ (} \langle \mathit{arrow}\rangle^{\star} \textbf{))} \\
\langle \mathit{opt}\rangle &\rightarrow \textbf{(} \langle \mathit{unlabeledCluster}\rangle \textbf{)} \mid \textbf{(} \langle \mathit{innerBox}\rangle?\ \langle \mathit{dashedClstrCases}\rangle? \textbf{)} \\
\langle \mathit{unlabeledCluster}\rangle &\rightarrow \textbf{(cluster dashed ( ) ( ))} \\
\langle \mathit{innerBox}\rangle &\rightarrow \textbf{(innerBox } \langle \mathit{solidClstrCases}\rangle^{+} \textbf{)} \\
\langle \mathit{dashedBoxCases}\rangle &\rightarrow \textbf{(cases } \langle \mathit{dashedBox}\rangle^{+} \textbf{)} \\
\langle \mathit{dashedBox}\rangle &\rightarrow \textbf{(box } \langle \mathit{classLabel}\rangle \textbf{ (} \langle \mathit{snglDashedCluster}\rangle \textbf{) ( ) (} \langle \mathit{arrow}\rangle^{\star} \textbf{))} \\
\langle \mathit{snglDashedCluster}\rangle &\rightarrow \textbf{(cases (cluster dashed } \langle \mathit{classLabel}\rangle \textbf{ ( )))} \\
\langle \mathit{dashedClstrCases}\rangle &\rightarrow \textbf{(cases } \langle \mathit{dashedCluster}\rangle^{+} \textbf{)} \\
\langle \mathit{dashedCluster}\rangle &\rightarrow \textbf{(cluster dashed } \langle \mathit{classLabel}\rangle \textbf{ (} \langle \mathit{arrow}\rangle^{\star} \textbf{))}
\end{aligned}
$$

Fig. 6. Abstract, formal syntax for $\mathcal{ALCN}$ model outlines

Clusters of *solid* diamonds represent individuals that must exist. In Fig. 4, it is mandatory that the graduate courses in question have as lecturer at least one tenured department professor holding a CompSci or Math PhD degree and supervising at least one graduate student from a total of 2 or more individuals. Likewise, the attendance must include students and graduate students.

Clusters of *dashed* diamonds represent optional individuals. If the cluster is labeled or has outgoing arrows, the individuals must belong to the corresponding class (e.g., "*Guest*" in Fig. 4). If the cluster is unlabeled, the individuals may belong to any class, subject to the constraints stipulated by the label and the outgoing arrows of the outer box where the cluster is located (e.g., in Fig. 4, the unlabeled cluster in the "*hasLecturer*" box represents lecturers that do not have to be tenured department professors, but that must hold a CompSci or Math PhD degree).

As indicated in the previous remark, *box labels and arrows originating from boxes* represent constraints that must be satisfied by all individuals corresponding to clusters in the box. In Fig. 4, all individuals attending the graduate courses in question must belong to class "*Enrolled*".

The *absence of a dashed cluster* in a box means that all the individuals represented in the box must belong to the classes specified by their respective labels and to the class specified by the box label and arrows (if present). This is evident in Fig. 4, where it is required that the lecturers hold a PhD degree *only* in CompSci or Math (a rather exclusivist and unfair requirement, but this is only an example).

Dashed boxes, always the target of dashed arrows, always contain a dashed cluster, representing optional individuals. In Fig. 4, the graduate courses in question may or may not involve the use of (up to 2) department labs.

"Among which" inner boxes contain clusters representing individuals that belong to subclasses of one or more classes specified in the outer box. In Fig. 4, the attendance of the graduate courses in question consists of students, some of which are required to be graduate students. Optionally, guests may attend.

The above remarks are included here only for pedagogical purposes. In fact, we define the precise semantics of model outlines by means of algorithm Descr, which, when given a model outline C (in abstract syntax), yields the $\mathcal{ALCN}$ concept description taken as the meaning of C. Algorithm Descr calls BoxDescr to build the concept description denoted by a box. Fig. 7 shows both algorithms.

The reader should refer to the grammar in Fig. 6 for the structure of the lists that the algorithms manipulate. These algorithms can be modified to produce more legible output; here, their only purpose is to serve as the precise semantics of model outlines. When given as input the model outline in Fig. 2, e.g., algorithm Descr returns the following description, which is equivalent to (1):

$$\begin{aligned}&\bot \sqcup \{\top \sqcap \forall \mathit{hasChild}.(\bot \sqcup \bot \sqcup \top) \sqcap \top \sqcap \exists \mathit{hasChild}.(\bot \sqcup \top)\\&\sqcap \forall \mathit{hasChild}.[\top \sqcap (\bot \sqcup \forall \mathit{hasChild}.(\bot \sqcup \bot \sqcup (\mathit{Doctor} \sqcap \neg \mathit{Lawyer})))]\}\end{aligned}$$

4 Constructing Model Outlines

We have presented elsewhere [10] detailed algorithms for translating $\mathcal{ALCN}$ concept descriptions into model outlines. Here, we incorporate some changes to the algorithms (e.g., to account for labeled optional clusters) and give a more informal explanation of the main steps involved in such a translation, using as a working example the concept description that originated the model outline in Fig. 4.

Given an $\mathcal{ALCN}$ concept description C, we start by converting C to modified disjunctive normal form (mDNF), applying simplification rules in the process. A concept description is in mDNF if it fits the pattern

$$D_1 \sqcup \ldots \sqcup D_n$$

where each disjunct D_i is a conjunction of the form

$$C_1 \sqcap \cdots \sqcap C_p$$

where each conjunct C_j is either a literal, or a collection of "intervals" of natural numbers (whose upper bound may be ∞) associated to a role R, or a description of the form $\forall R.C'$ or of the form $\exists R.C'$, where C' is itself in mDNF.

The modification is in the way number restrictions are represented: using appropriate rewrite rules, any conjunction of cardinality restrictions over a role R_i can be converted to a collection of "intervals" of natural numbers.[2]

[2] For role R, the interval $[m, n]$ represents the constraint $(\geq m.R \sqcap \leq n.R)$. Likewise, $[m, m]$ represents $(= m.R)$, and $[0, m]$ represents $(\leq m.R)$, and $[m, \infty]$ represents $(\geq m.R)$.

```
DESCR($C$)                          ▷ $C$ has the form ( cases $C_1 \cdots C_m$ )
1  $Descr \leftarrow \bot$
2  for each $C_i$ in $C_1, \ldots, C_m$      ▷ $C_i$ has the form ( cluster $S\ L$ ( $A_1 \cdots A_n$ ) )
3        do if $L =$ ( )
4             then $Case \leftarrow \top$
5             else $Case \leftarrow L$
6           for each $A_j$ in $A_1, \ldots, A_n$
7               do $Case \leftarrow Case \sqcap$ BOXDESCR($A_j$)
8           $Descr \leftarrow Descr \sqcup Case$
9  return $Descr$
```

```
BOXDESCR($A$)    ▷ $A$ has the form ( arrow $S\ RN$ ( $I_1 \cdots I_n$ ) ( cases $B_1 \cdots B_m$ ) )
 1  $BDescr \leftarrow \bot$
 2  if $n = 0$                                   ▷ No cardinality restrictions
 3     then $Card \leftarrow \top$
 4     else $Card \leftarrow \bot$
 5          for each $I_j$ in $I_1, \ldots, I_n$          ▷ $I_j$ is "interval" of the form ( $X\ Y$ )
 6              do if $Y =$ infty
 7                    then $Card \leftarrow (Card \sqcup \geq X.RN)$
 8                    else $Card \leftarrow (Card \sqcup (\geq X.RN \sqcap \leq Y.RN))$
 9  for each $B_i$ in $B_1 \cdots B_m$
10     ▷ Each box case $B_i$ has the form ( box $BL$ ( $C_1 \cdots C_p$ ) $Opt$ ( $A'_1 \cdots A'_q$ ) )
11       do $Universal \leftarrow \bot$; $Existentials \leftarrow \top$
12          for each $C_j$ in $C_1, \ldots, C_p$                  ▷ Cluster cases
13              do $Universal \leftarrow Universal \sqcup$ DESCR($C_j$)
14                 if $C_j$ has the form ( cluster solid ... )
15                    then $Existentials \leftarrow Existentials \sqcap \exists RN$.DESCR($C_j$)
16          if $Opt$ contains ( innerBox $C'_1 \cdots C'_r$ )
17             then for each $C'_j$ in $C'_1, \ldots, C'_r$
18                     do $Existentials \leftarrow Existentials \sqcap \exists RN$.DESCR($C'_j$)
19          if $Opt$ contains ( cluster dashed ( ) ( ) )
20             then $Universal \leftarrow \top$
21          if $Opt$ contains ( cases $C''_1 \cdots C''_s$ )          ▷ Optional clusters
22             then for each $C''_j$ in $C''_1, \ldots, C''_s$
23                     do $Universal \leftarrow Universal \sqcup$ DESCR($C''_j$)
24          $Universal \leftarrow \forall RN.(Universal)$
25          if $BL =$ ( )
26             then $BCase \leftarrow Universal \sqcap Existentials$
27             else $BCase \leftarrow \forall RN.BL \sqcap Universal \sqcap Existentials$
28          for each $A'_j$ in $A'_1, \ldots, A'_q$                   ▷ Box arrows
29              do $BCase \leftarrow BCase \sqcap \forall RN$.BOXDESCR($A'_j$)
30          $BDescr \leftarrow BDescr \sqcup BCase$
31  return $Card \sqcap BDescr$
```

Fig. 7. Algorithms to convert from model outlines to $\mathcal{ALCN}$

To each D_i we then apply the simplification rule

$$\forall R.C_1 \sqcap \ldots \sqcap \forall R.C_n \; \triangleright \; \forall R.(C_1 \sqcap \ldots \sqcap C_n)$$

As a result, we obtain C', which is a disjunction $D'_1 \sqcup \ldots \sqcup D'_n$, where each D'_i can be written as

$$L_1 \sqcap \ldots \sqcap L_m \sqcap C_1 \sqcap \cdots \sqcap C_p$$

where each L_i is a literal, and each C_j can be written as

$$\forall R.F \sqcap \exists R.G_1 \sqcap \ldots \sqcap \exists R.G_q \sqcap K$$

where F and all the G_i are in mDNF and K is a collection of intervals of natural numbers representing cardinality restrictions over role R. Any (or all) of these elements may be absent. Note that we have grouped the conjuncts according to the role R they refer to. Later, when the model outline is built, each of these groups will originate an arrow labeled by R.

Following these guidelines, the simplified mDNF of the concept description corresponding to the example model outline in Fig. 4 is found to be

$$\begin{aligned}
&\mathit{GraduateCourse} && \text{(2a)}\\
&\sqcap \forall \mathit{hasLecturer}. && \text{(2b)}\\
&\quad [\forall \mathit{holdsPhDIn}.(\mathit{CompSci} \sqcup \mathit{Math}) && \text{(2c)}\\
&\quad \sqcap \exists \mathit{holdsPhDIn}.(\mathit{CompSci} \sqcup \mathit{Math})] && \text{(2d)}\\
&\sqcap \exists \mathit{hasLecturer}.(\mathit{DeptProfessor} \sqcap \mathit{Tenured} \sqcap \exists \mathit{supervises}.\mathit{GradStudent} && \text{(2e)}\\
&\qquad\qquad \sqcap \{[2,\infty]\}.\mathit{supervises}) && \text{(2f)}\\
&\sqcap \{[2,2]\}.\mathit{hasLecturer} && \text{(2g)}\\
&\sqcap \forall \mathit{hasAttendance}.[(\mathit{Student} \sqcap \mathit{Enrolled}) \sqcup (\mathit{Guest} \sqcap \mathit{Enrolled})] && \text{(2h)}\\
&\sqcap \exists \mathit{hasAttendance}.\mathit{Student} && \text{(2i)}\\
&\sqcap \exists \mathit{hasAttendance}.\mathit{GradStudent} && \text{(2j)}\\
&\sqcap \{[10,50]\}.\mathit{hasAttendance} && \text{(2k)}\\
&\sqcap \forall \mathit{usesLab}.(\mathit{DeptLab} \sqcap \neg \mathit{Closed}) && \text{(2l)}\\
&\sqcap \{[0,2]\}.\mathit{usesLab} && \text{(2m)}
\end{aligned}$$

Note how the constraints have been grouped by the roles they act upon. Note also how the cardinality constraints in lines (2f), (2g), (2k) and (2m) have been written with (singleton) collections of intervals of natural numbers.

Two transformations must be effected before the model outline can be built.

The first one concerns lines (2b)–(2d), where the set of objects related to the lecturers through *holdsPhDIn* is *closed*: i.e., the lecturers must hold *some* PhD degree in CompSci or Math and *only* PhD degrees in CompSci or Math.

The algorithm detects such a closure whenever it finds conjuncts of the form

$$\forall R(C_1 \sqcup \cdots \sqcup C_n) \sqcap \exists R.C_1 \sqcap \cdots \sqcap \exists R.C_n$$

Here, we have $n = 1$ and $C_1 = \mathit{CompSci} \sqcup \mathit{Math}$. Then, to indicate the closure, the algorithm refrains from adding a dashed, unlabeled cluster to the target box of the *holdsPhDIn* arrow (see Fig. 4).

The second transformation is similar: in lines (2h)–(2j), we can see there is some sort of closure related to the role *hasAttendance*, but the situation is more complicated. In fact, this is the general case, which also includes the first transformation. Whenever the conjuncts for role R are of the form

$$\begin{aligned}&\forall R[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D) \sqcup (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)]\\ &\sqcap \exists R.C_1 \sqcap \cdots \sqcap \exists R.C_n \sqcap \exists R.F_1 \sqcap \cdots \sqcap \exists R.F_q\end{aligned}$$

where D is a conjunction (with $D = \top$ as the trivial case) it proceeds as follows:

- Solid clusters for $C_1, \ldots, C_n$ are created in the main target box for the R-arrow.
- The main target box for the R-arrow gets D as a label. If $D = \top$, this label is not shown.
- The main target box for the R-arrow gets an “among which” inner box containing solid clusters for $F_1, \ldots, F_q$.
- Dashed clusters for $C_{n+1}, \ldots, C_p$ are created in the main target box for the R-arrow.

In our example description, in lines (2h)–(2j), we have that $n = 1$, and $C_1 = \mathit{Student}$, and $D = \mathit{Enrolled}$, and $p = 1$, and $C_2 = \mathit{Guest}$, and $q = 1$, and $F_1 = \mathit{GradStudent}$.

5 Evaluation

We have conducted a usability study in order to evaluate our proposed diagrammatic notation. The main aim was to *test the usefulness of model outlines for the understanding of complex concept descriptions.*

Note that it is the model outline *notation* itself that is being evaluated, not a specific graphical user interface (GUI) implementing the notation. Thus, the focus of the study is on understanding, not on interaction. We find this to be an advantage, as changes can be made to the notation before we are committed to a specific GUI, and problems can be identified in relation to specific features of the notation, so that special attention can be given to these problems in order to solve or mitigate them through the use of appropriate human interaction techniques. From a practical point of view, this potentially reduces the need for radical, costly changes after implementation.

Likewise, we have chosen model outlines for the simpler $\mathcal{ALCN}$ language so we could find out early if something needs to be changed in our most basic assumptions. The result of this test will help us design the extensions of model outlines to deal with more expressive concept languages.

Following [14], we defined our *main goal* as: *Model outlines can help users with little or no training in Logic to understand complex concept descriptions. In particular, model outlines are more effective than Manchester OWL for this task.*

Manchester OWL (see Fig. 3 and also [4]) is a textual notation for DL which uses keywords for logical symbols (e.g., "**SOME**" for "$\exists$"), infix notation for restrictions (e.g., "*hasChild* **SOME** *Man*" for "$\exists hasChild.Man$"), syntax highlighting and indentation in order to make descriptions more readable for nonspecialists. So, we are comparing our diagrammatic notation with a textual notation designed for the same target audience. (As the test participants were all Brazilians, we used Portuguese translations of the Manchester OWL keywords.)

Next, we defined a set of *concerns*, in the form of questions like: *Can users understand the meaning of X?*, where X is one of the elements present in model outlines (solid clusters, dashed clusters, arrows, boxes, inner boxes, case widgets, etc.). Specific concerns were also formulated (e.g., "Can users understand that individuals in "among which" inner boxes are mandatory?").

We selected 10 participants for our study ([14] recommends 6 to 12). These participants come from several backgrounds and occupations, as detailed below. All received detailed information on the procedures and on their rights as participants. All signed terms of informed consent.

One session of the study consisted of the following activities: a pre-test questionnaire, a tutorial on notation A, a specification on domain X using notation A, 15 questions, a post-task questionnaire, a tutorial on notation B, a specification on domain Y using notation B, 15 questions, a post-task questionnaire, and a post-test questionnaire. Notations A and B alternated between model outlines and Manchester OWL. Domains X and Y alternated between graduate courses (which included Fig. 4 of this paper) and family relations. Each participant answered 15 questions for each domain. The questions for each domain were fixed, regardless of the notation used. For each domain, half the participants answered questions on model outlines, and half answered questions on Manchester OWL specifications. Half the participants saw model outlines before Manchester OWL, and half saw Manchester OWL before model outlines.

The number of correct answers and the time to answer were measured. Additional information was obtained in the form of comments collected through the "thinking out loud" protocol [14] and through questionnaires. Table 1 shows the occupation and the number of correct answers for each participant:

For the graduate courses domain, we note the following highlights:

Question 8 was related to Fig. 4 of this paper, and elicited *5 errors* using Manchester OWL, and *no errors* using model outlines. The question was: "If a course is attended only by students that are *not* graduate students, does the course meet the specification?" The error was probably induced by the abbreviation recommended in [4]: "*hasAttendance* **SOME** [*Student*, *GradStudent*]", which seems to have evoked the idea that the bracketed list consisted of a set of alternatives. This question was answered correctly by all participants using model outlines, which indicates that users understood the meaning of "among which" inner boxes.

Question 14 elicited *4 errors* using Manchester OWL, and *3 errors* using model outlines. This question was about a specification consisting of 4 cases. The situation proposed in the question satisfied exactly one of the 4 cases. With

Table 1. Occupation and number of correct answers for each participant

Occupation	**Correct answers (model outlines)**	**Correct answers (Manchester OWL)**
Logician	15	14
Theoretical physicist	15	12
Software engineer	15	12
Secretary	15	10
Nurse	13	12
Graphics designer	13	12
Social worker	13	11
Comp. Science undergrad	13	10
Production engineer	13	9
Mathematician	12	14
Totals:	137	116
Percentages:	91.3%	77.3%

Manchester OWL, the participants had difficulty in finding their way among multiple parentheses and complex disjunctions. With model outlines, they apparently thought that the proposed situation had to satisfy *all* cases.

For the family relations domain, we note the following highlights:

Question 6 elicited *4 errors* using Manchester OWL, and *no errors* using model outlines. This question asked if a person satisfying the given specification could have jobless children. The specification in Manchester OWL included the sentence "*hasChild* **SOME** (*Man* **AND** *worksAt* **ONLY** *Hospital*)". Apparently, the users forgot that "ONLY" (which stands for "$\forall$") does not imply the existence of objects. In the model outline, the presence of a dashed cluster, a dashed box and a dashed arrow made it clear that existence was not required.

Question 8 elicited *3 errors* using Manchester OWL, and *1 error* using model outlines. This question asked if a person satisfying the given specification had to have a grandchild working as a surgeon. Some users found it confusing to follow the composition of roles (*hasChild–hasChild*), and were again, as in question 8 about graduate courses, confused by the Manchester OWL abbreviation "**SOME** $[\cdots]$". In the model outline, the presence of a solid cluster labeled *Surgeon* inside an "among which" inner box made the correct answer more clear.

One trend was clearly observed in both domains: specifications that involve cases (i.e., complex disjunctions), such as the one in Fig. 5 of this paper, are more difficult to understand than those that do not, as Table 2 indicates.

Among the comments offered by the participants, many indicated confusion due to the way cases were presented in model outlines (like in Fig. 5 of this paper, the layout consisted of 4 diagrams on a single page). Some users thought that all 4 diagrams had to be satisfied. This is clearly one weakness of model outlines

Table 2. Number of correct answers per domain and type of question

Domain and type of question	**Correct answers (model outlines)**	**Correct answers (Manchester OWL)**
Family, no cases	94%	72%
Courses, no cases	96%	84%
Family, with cases	84%	72%
Courses, with cases	84%	80%

(on paper) that we must try to eliminate in the GUI implementation. We predict that such confusion will not arise if the user interacts with the model outline (e.g., dynamically expanding and collapsing cases). The GUI should also make clear when clusters in different cases actually correspond to the same cluster, by showing one single cluster which can be expanded in different ways.

As for time: in the courses domain, each user took in average 28 seconds per question, regardless of the notation. In the family relations domain, each user took in average 26 seconds per question with model outlines, but 40 seconds per question with Manchester OWL.

Of the 10 participants, 5 said they preferred model outlines, 4 said they liked both notations equally well, and 1 said both notations were equally bad.

6 Conclusions

The main achievements of the work related here are the reformulation of our model outline notation and the results of our first usability test, comparing model outlines to Manchester OWL.

Ontology visualization is a very active field of study. The survey [6] discusses over 40 ontology visualization tools, all of them developed in the past 10 years. All of those tools are *general*, in the sense that they use one single visualization framework to show several types of information about the ontology: the subsumption hierarchy, roles, etc. In particular, those tools show concept descriptions either textually (e.g., Protégé) or in the form of abstract syntax trees (as in Fig. 1 of this paper).

Model outlines, on the other hand, are *specialized*, having been designed specifically to show concept descriptions. Although the notation used is new, our usability test indicates it is intuitive enough to be understood by nonspecialists. The specialized nature of model outlines suggests that they can be *integrated* with a more general tool, so that users can easily switch views, e.g., from the subsumption hierarchy as a tree to the definition of a class as a model outline.

We are currently implementing a concept description browser based on model outlines, as a Protégé plugin. We are taking special care to rely on graphical conventions and interaction techniques that profit from the vast body of knowledge related to visual perception and cognitive principles, as described, e.g., in [15].

Work is also under way to extend model outlines to the concept language associated to OWL 2 [13].

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook, 2nd edn. Cambridge University Press, Cambridge (2007)
2. McGuinness, D.L., da Silva, P.P.: Explaining answers from the semantic web: the inference web approach. Journal of Web Semantics 1(4), 397–413 (2004)
3. Bailey, J., Bry, F., Furche, T., Schaffert, S.: Web and semantic web query languages: A survey. In: Eisinger, N., Maluszynski, J. (eds.) Reasoning Web. LNCS, vol. 3564, pp. 35–133. Springer, Heidelberg (2005)
4. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wan, H.: The Manchester OWL syntax. In: OWL: Experiences and Directions (2006)
5. Fuchs, N.E., Kaljurand, K., Schneider, G.: Attempto Controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In: FLAIRS 2006 (2006)
6. Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., Giannopoulou, E.: Ontology visualization methods—a survey. ACM Comput. Surv. 39(4), Article 10 (2007)
7. Gaines, B.R.: Designing visual languages for description logics. Journal of Logic, Language and Information 18(2), 217–250 (2009)
8. Borgida, A., Franconi, E., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: Explaining ALC subsumption. In: International Workshop on Description Logics (DL'99), Linköping, Sweden (1999)
9. Krivov, S., Williams, R., Villa, F.: GrOWL: A tool for visualization and editing of OWL ontologies. Journal of Web Semantics 5(2), 54–57 (2007)
10. do Amaral, F.N., Bazílio, C.: Visualization of Description Logic models. In: The 21st International Workshop on Description Logics (DL 2008), Dresden, Germany (2008), http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-353/NaufelMartins.pdf
11. do Amaral, F.N.: Visualizing the semantics (not the syntax) of concept descriptions. In: Proceedings of VI TIL, Vila Velha, ES (2008), http://www.nilc.icmc.usp.br/til/til2008/p336-do_amaral.pdf
12. Howse, J., Molina, F., Shin, S.J., Taylor, J.: On diagram tokens and types. In: Hegarty, M., Meyer, B., Narayanan, N.H. (eds.) Diagrams 2002. LNCS (LNAI), vol. 2317, pp. 146–160. Springer, Heidelberg (2002)
13. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press, Menlo Park (2006)
14. Dumas, J., Redish, J.: A Practical Guide to Usability Testing. Intellect (1994)
15. Ware, C.: Information Visualization: Perception for Design. Morgan Kaufmann Publishers Inc., San Francisco (2004)

A Rule-Based Language for Complex Event Processing and Reasoning

Darko Anicic[1], Paul Fodor[2], Sebastian Rudolph[3], Roland Stühmer[1], Nenad Stojanovic[1], and Rudi Studer[1]

[1] FZI Research Center for Information Technology, University of Karlsruhe, 76131, Germany
[2] State University of New York at Stony Brook, USA
[3] Institut AIFB, University of Karlsruhe, Karlsruhe, Germany

Abstract. Complex Event Processing (CEP) is concerned with timely detection of complex events within multiple streams of atomic occurrences. It has useful applications in areas including financial services, mobile and sensor devices, click stream analysis etc. Numerous approaches in CEP have already been proposed in the literature. Event processing systems with a logic-based representation have attracted considerable attention as (among others reasons) they feature formal semantics and offer reasoning service. However logic-based approaches are not optimized for run-time event recognition (as they are mainly query-driven systems). In this paper, we present an expressive logic-based language for specifying and combining complex events. For this language we provide both a syntax as well as a formal declarative semantics. The language enables efficient run time event recognition and supports deductive reasoning. Execution model of the language is based on a compilation strategy into Prolog. We provide an implementation of the language, and present the performance results showing the competitiveness of our approach.

1 Introduction

Recently there has been made a significant paradigm shift toward *real-time* computing in the research, as well as, in industry. Databases and data warehouses are about looking what happened in the past. On the other hand, Complex Event Processing (CEP) is about processing real-time events, i.e., about detecting what has just happened or what is about to happen.

An *event* represents something that occurs, happens or changes the current state of affairs. For example, an event may signify a problem or an impending problem, a threshold, an opportunity, an information becoming available, a deviation etc. The general task of CEP can be described as follows. Within some dynamic setting, events take place. Those *atomic events* are instantaneous, i.e., they happen at one specific point in time and have a duration of zero. Notifications about these occurred events together with their timestamps and possibly further associated data (such as involved entities, numerical parameters of the event, or provenance data) enter the CEP system in the order of their occurrence.

The CEP system further features a set of *complex event descriptions*, by means of which *complex events* can be specified as temporal constellations of atomic events. The complex events thus defined can in turn be used to compose even more complex events and so forth. As opposed to atomic events, those complex events are not considered instantaneous but are endowed with a time *interval* denoting when the event started and when it ended.

The purpose of the CEP system is now to detect complex events within this input stream of atomic events. That is, the system is supposed to notify that the occurrence of a certain complex event has been detected, as soon as the system is notified of an atomic event that completes a sequence which makes up the complex event due to the complex event description. This notification may be accompanied by additional information composed from the atomic events' data. As a consequence of this detection (and depending on the associated data), responding actions can be taken, yet this is outside the scope of this paper.

Our approach for CEP is based on declarative (logic) rules. It has been shown elsewhere [13,16,15,2,7,12,17] that logic-based approaches for event processing have various advantages. First, they are *expressive* enough and convenient to represent diverse complex event patterns. They come with a *formal declarative* semantics. Moreover declarative rules are free of side-effects (e.g. confluence problem). Second, integration of *query processing* with event processing is easy and natural (e.g. processing of *recursive* queries). Third, our experience with use of logic rules in implementation of the main constructs in CEP as well as in providing extensibility of a CEP system is very positive and encouraging (e.g. number of code lines in logic programming is significantly smaller than in procedural programming). Ultimately, a logic-based event model allows for *reasoning* over events, their relationships, entire state, and possible *contextual knowledge* available for a particular domain. Reasoning about *temporal* knowledge (i.e., events) and *static* or *evolving* knowledge (i.e., facts, rules and ontologies) is a feature beyond of the state-of-the-art in CEP [1,6,14].

Apart from the above mentioned strengths, event processing systems [13,16,15,12,17] based on various logic formalism have some shortcomings too. One significant shortcoming is *data* or *event-driven* computation. Deductive systems are rather suited for a *request-response* computation. That is, for given a *request*, an inference engine will evaluate available knowledge (i.e. rules and facts) and *respond* with an answer. This means that the event inference engine needs to check if this pattern can be deduced or not. The check is performed at the time when such a request is posed. If satisfied by the time when the request is processed, a complex event will be reported. If not, the pattern is not detected until the next time the same request is processed (though it can become satisfied in-between the two checks). Contrary to this, event processing demands *data-driven* computation (as handled by various approaches such as non-deterministic finite automata (NFA) [1], Petri Nets [11], RETE algorithm [10] etc.). Unfortunately approaches grounded on NFA and Petri Nets do not feature reasoning capabilities; and RETE based approaches may be integrated with deductive rules [4] but have difficulties to handle aggregates over event streams, and to implement different event consumption policies [8].

[17] follows the mentioned request-response (or so called *query-driven*[1]) approach. It proposes to define queries that are processed repetitively at given intervals, e.g. every

[1] If a request is represented as a query (what is a usual case).

10 seconds, trying to discover new events. However, generally events are not periodic or if so might have differing periods and nevertheless complex events should be detected as soon as they occur (not in a predefined time window). To overcome this issue, in [7], an incremental evaluation was proposed. The approach is aimed at avoiding redundant computations (particularly re-computation of joins) every time a new event arrives. The authors suggest to utilize relational algebra evaluation techniques such as incremental maintenance of materialized views.

Our language for CEP, ETALIS, is developed to close the gap between event-driven and logic-based systems. We present a *rule-based language* with a clear syntax and a declarative formal semantics. The language is powerful enough to effectively *express* and *evaluate* all thirteen Allen's temporal relationships [3]. Unlike other non-logic-based CEP languages [1,11], our language features *inference* capabilities; and unlike other logic-based approaches, it has a different execution model that compiles complex event patterns into logic rules and enables timely, *event-driven* detection of complex events. Finally unlike RETE-based approaches, recursive rules of our language enable processing of unbound event streams and applying aggregation functions on them; yet recursive rules are out of scope of this paper. The contribution also includes an *implementation* of the language, and experimental results of our evaluation.

2 Rule-Based Language for Event Processing and Reasoning

2.1 Syntax

In this section we present the formal syntax of the our language for event processing, while in the remaining sections of the paper, we will gradually introduce other aspects of the language (i.e. the declarative semantics and run-time detection of complex events as well as the performance of a prototype based on the language[2]).

The syntax of the our language allows for the description of *time* and *events*. We represent time instants as well as durations as nonnegative rational numbers $q \in \mathbb{Q}^+$. Events can be atomic or complex. An *atomic event* refers to an instantaneous occurrence of interest. Atomic events are expressed as ground atoms (i.e. predicates followed by arguments which are terms not containing variables). Intuitively, the arguments of a ground atom describing an atomic event denote information items (i.e. event data) that provide additional information about that event.

Atomic events can be composed to form *complex events* via *event patterns*. We use event patterns to describe how events can (or have to) be temporally situated to other events or absolute time points. The language P of event patterns is formally defined by

$$\begin{aligned} P ::= \texttt{pr}(t_1, \ldots, t_n) \mid P \text{ WHERE } t \mid q \mid (P).q \\ \mid P \text{ BIN } P \mid \text{NOT}(P).[P, P] \end{aligned}$$

Thereby, $\texttt{pr}$ is a predicate name with arity n, t_i denote terms, t is a term of type boolean, q is a nonnegative rational number, and BIN is one of the binary operators SEQ, AND,

[2] Our prototype, ETALIS, is an open source project, available at: `http://code.google.com/p/etalis`

PAR, OR, EQUALS, MEETS, DURING, STARTS, or FINISHES. As a side condition, in every expression p WHERE t, all variables occurring in t must also occur in pattern p.

Finally, an *event rule* is defined as a formula of the shape

$$\texttt{pr}(t_1, \ldots, t_n) \leftarrow p$$

where p is an event pattern containing all variables occurring in $\texttt{pr}(t_1, \ldots, t_n)$.

After introducing the formal syntax of our formalism, we will give some examples to provide some intuitive understanding before proceeding with the formal semantics in the next section. Adhering to a stock market scenario, one instantaneous event (not requiring further specification) might be $\texttt{market\_closes}()$. Other events with additional information associated via arguments would be $\texttt{bankrupt}(lehman)$ or $\texttt{buys}(citigroup, wachovia)$. Within patterns, variables instead of constants may occur as arguments, whence we can write $\texttt{bankrupt}(X)$ as a pattern matching all bankruptcy events irrespective of the victim. "Artificial" time-point events can be defined by just providing the according timestamp.

Figure 1 demonstrates the various ways of constructing complex event descriptions from simpler ones in our language for event processing. Moreover, the figure informally introduces the semantics of the language, which will further be defined in Section 2.2.

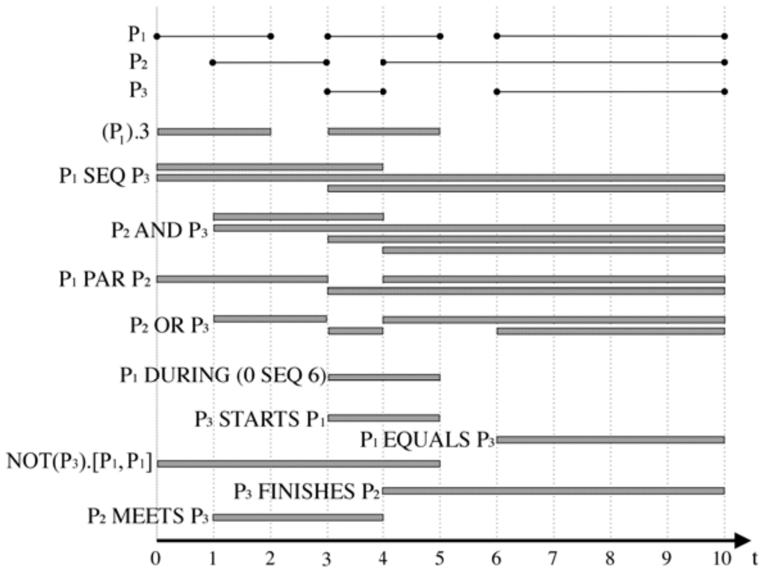


Fig. 1. Language for Event Processing - Composition Operators

Let us assume that instances of three complex events, P_1, P_2, P_3, are occurring in time intervals as shown in Figure 1. Vertical dashed lines depict different time units, while the horizontal bars represent detected complex events for the given patterns. In the following, we give the intuitive meaning for all patterns from the figure:

- $(P_1).3$ detects an occurrence of P_1 if it happens within an interval of length 3.
- P_1 SEQ P_3 represents a sequence of two events, i.e. an occurrence of P_1 is followed by an occurrence of P_3; thereby P_1 must end before P_3 starts.
- P_2 AND P_3 is a pattern that is detected when instances of both P_2 and P_3 occur no matter in which order.
- P_1 PAR P_2 occurs when instances of both P_2 and P_3 happen, provided that their intervals have a non-zero overlap.
- P_2 OR P_3 is triggered for every instance of P_2 or P_3.
- P_1 DURING $(0$ SEQ $6)$ happens when an instance of P_1 occurs during an interval; in this case, the interval is built using a sequence of two atomic time-point events (one with $q = 0$ and another with $q = 6$, see the syntax above).
- P_1 EQUALS P_3 is triggered when the two events occur exactly at the same time interval.
- NOT$(P_3).[P_1, P_1]$ represents a negated pattern. It is defined by a sequence of events (delimiting events) in the square brackets where there is no occurrence of P_3 in the interval. In order to invalidate an occurrence of the pattern, an instance of P_3 must happen in the interval formed by the end time of the first delimiting event and the start time of the second delimiting event. In this example delimiting events are just two instances of the same event, i.e. P_1. Different treatments of negation are also possible, however we adopt one from [8].
- P_3 STARTS P_1 is detected when an instance of P_3 starts at the same time as an instance of P_1 but ends earlier.
- P_3 FINISHES P_2 is detected when an instance of P_3 ends at the same time as an instance of P_1 but starts later.
- P_2 MEETS P_3 happens when the interval of an occurrence of P_2 ends exactly when the interval of an occurrence of P_3 starts.

It is worth noting that the defined pattern language captures the set of all possible 13 relations on two temporal intervals as defined in [3]. The set can also be used for rich temporal reasoning.

2.2 Declarative Semantics

We define the declarative formal semantics of our language for event processing in a model-theoretic way.

Note that we assume a fixed interpretation of the occurring function symbols, i.e. for every function symbol f of arity n, we presume a predefined function $f^* : Con^n \to Con$. That is, in our setting, functions are treated as built-in utilities.

As usual, a *variable assignment* is a mapping $\mu : Var \to Con$ assigning a value to every variable. We let μ^* denote the extension of μ to terms defined in the usual way:

$$\mu^* : \begin{cases} v \mapsto \mu(v) & \text{if } v \in Var, \\ c \mapsto c & \text{if } c \in Con, \\ f(t_1, \dots, t_n) \mapsto f^*(\mu^*(t_1), \dots, \mu^*(t_n)) & \text{otherwise.} \end{cases}$$

In addition to the set of rules $\mathcal{R}$, we fix an *event stream*. The event stream is formalized as a mapping $\epsilon : Ground \to 2^{\mathbb{Q}^+}$ from ground predicates into sets of nonnegative rational numbers. It thereby indicates at what time instants what elementary events occur. As a side condition, we require ϵ to be free of accumulation points, i.e. for every $q \in \mathbb{Q}^+$, the set $\{q' \in \mathbb{Q}^+ \mid q' < q \text{ and } q' \in \epsilon(g) \text{ for some } g \in Ground\}$ is finite.

pattern	$\mathcal{I}_\mu$(pattern)
$\texttt{pr}(t_1,\ldots,t_n)$	$\mathcal{I}(\texttt{pr}(\mu^*(t_1),\ldots,\mu^*(t_n)))$
p WHERE t	$\mathcal{I}_\mu(p)$ if $\mu^*(t) = \mathit{true}$ $\emptyset$ otherwise.
q	$\{\langle q,q\rangle\}$ for all $q\in\mathbb{Q}^+$
$(p).q$	$\mathcal{I}_\mu(p)\cap\{\langle q_1,q_2\rangle \mid q_2-q_1=q\}$
p_1 SEQ p_2	$\{\langle q_1,q_4\rangle \mid \langle q_1,q_2\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle\in\mathcal{I}_\mu(p_2)$ and $q_2<q_3\}$
p_1 AND p_2	$\{\langle \mathsf{min}(q_1,q_3),\mathsf{max}(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle\in\mathcal{I}_\mu(p_2)\}$
p_1 PAR p_2	$\{\langle \mathsf{min}(q_1,q_3),\mathsf{max}(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle\in\mathcal{I}_\mu(p_2)$ and $\mathsf{max}(q_1,q_3)<\mathsf{min}(q_2,q_4)\}$
p_1 OR p_2	$\mathcal{I}_\mu(p_1)\cup\mathcal{I}_\mu(p_2)$
p_1 EQUALS p_2	$\mathcal{I}_\mu(p_1)\cap\mathcal{I}_\mu(p_2)$
p_1 MEETS p_2	$\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_2,q_3\rangle\in\mathcal{I}_\mu(p_2)\}$
p_1 DURING p_2	$\{\langle q_3,q_4\rangle \mid \langle q_1,q_2\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle\in\mathcal{I}_\mu(p_2)$ and $q_3<q_1<q_2<q_4\}$
p_1 STARTS p_2	$\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle\in\mathcal{I}_\mu(p_2)$ and $q_2<q_3\}$
p_1 FINISHES p_2	$\{\langle q_1,q_3\rangle \mid \langle q_2,q_3\rangle\in\mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle\in\mathcal{I}_\mu(p_2)$ and $q_1<q_2\}$
NOT$(p_1).[p_2,p_3]$	$\mathcal{I}_\mu(p_2$ SEQ $p_3)\setminus\mathcal{I}_\mu(p_2$ SEQ p_1 SEQ $p_3)$

Fig. 2. Definition of extensional interpretation of event patterns. We use $p_{(x)}$ for patterns, $q_{(x)}$ for rational numbers, $t_{(x)}$ for terms and pr for event predicates.

Now, we define an interpretation $\mathcal{I} : \mathit{Ground} \to 2^{\mathbb{Q}^+\times\mathbb{Q}^+}$ as a mapping from the ground atoms to sets of pairs of nonnegative rationals, such that $q_1 \leq q_2$ for every $\langle q_1,q_2\rangle \in \mathcal{I}(g)$ for all $g \in \mathit{Ground}$.

Given an event stream ϵ, an interpretation $\mathcal{I}$ is called a *model* for a rule set $\mathcal{R}$ – written as $\mathcal{I} \models_\epsilon \mathcal{R}$ – if the following conditions are satisfied:

C1 $\langle q,q\rangle \in \mathcal{I}(g)$ for every $q \in \mathbb{Q}^+$ and $g \in \mathit{Ground}$ with $q \in \epsilon(g)$

C2 for every rule $\mathit{atom} \leftarrow \mathit{pattern}$ and every variable assignment μ we have $\mathcal{I}_\mu(\mathit{atom}) \subseteq \mathcal{I}_\mu(\mathit{pattern})$ where $\mathcal{I}_\mu$ is inductively defined as displayed in Fig. 2.

Given an interpretation $\mathcal{I}$ and some $q \in \mathbb{Q}^+$, we let $\mathcal{I}|_q$ denote the interpretation defined via $\mathcal{I}|_q(g) = \mathcal{I}(g) \cap \{\langle q1,q2\rangle \mid q2-q1 \leq q\}$.

Given two interpretations $\mathcal{I}$ and $\mathcal{J}$, we say that $\mathcal{I}$ is *preferred* to $\mathcal{J}$ if there exists a $q \in \mathbb{Q}^+$ with $\mathcal{I}|_q \subset \mathcal{J}|_q$.

A model $\mathcal{I}$ is called *minimal* if there is no other model preferred to $\mathcal{I}$. It is easy to show that for every event stream ϵ and rule set $\mathcal{R}$ there is a unique minimal model $\mathcal{I}^{\epsilon,\mathcal{R}}$.

Finally, given an atom a and two rational numbers q_1, q_2, we say that the event $a^{[q_1,q_2]}$ is a *consequence* of the event stream ϵ and the rule base $\mathcal{R}$ (written $\epsilon,\mathcal{R} \models a^{[q_1,q_2]}$), if $\langle q_1,q_2\rangle \in \mathcal{I}_\mu^{\epsilon,\mathcal{R}}(a)$ for some variable assignment μ.

It can be easily verified that the behavior of the event stream ϵ beyond the time point q_2 is irrelevant for determining whether $\epsilon,\mathcal{R} \models a^{[q_1,q_2]}$ is the case.[3] This justifies to

[3] More formally, for any two event streams ϵ_1 and ϵ_2 with $\epsilon_1(g) \cap \{\langle q,q'\rangle \mid q' \leq q_2\} = \epsilon_2(g) \cap \{\langle q,q'\rangle \mid q' \leq q_2\}$ we have that $\epsilon_1,\mathcal{R} \models a^{[q_1,q_2]}$ exactly if $\epsilon_2,\mathcal{R} \models a^{[q_1,q_2]}$.

take the perspective of ϵ being only partially known (and continuously unveiled along a time line) while the task is to detect event-consequences as soon as possible.

3 A Deductive Rule-Based Approach for Complex Event Processing

In Section 1 we have numbered few advantages of CEP approaches based on logic. The majority of state-of-the-art in CEP is however not based on logic rules [1,6,14]; hence these advantages can be seen as features going beyond the state-of-the-art. In this section we review the features once again, justifying the design principles of our proposed language.

Expressive, formal, and declarative semantics. In the previous section we have defined *formal* and *declarative* semantics of an event processing language. CEP is a real-time processing, involving very often multi-threading and distributed processing. In such an environment, *declarative* semantics guarantees *predictability* and *repeatability* of results produced by an event processing system. This is not case in procedural CEP languages where, e.g., for the same input stream and the same set of event patterns, the system may produce two different results. Our proposed semantics is also *expressive* enough to capture all thirteen Allen's temporal relationships [3].

To evaluate expressivity of our language in practice, we have implemented *Fast Flower Delivery* use case[4] from [9]. The use case describes distribution of flowers from multiple stores (in a large city). The distribution is handled by a group of drivers who need to accomplish their assignments in a timely fashion. In the remaining part of this section we will use some of the use case patterns to demonstrate capabilities of our language.

Database and rule queries. Database information may serve in *enriching* an event with additional data. For instance, whenever a customer purchases flowers an event $request$ is triggered. A delivery request event (dlv_req) consists of the initial event $request$, *enriched* by additional information. This information is taken from a database relation $store\_info$, and can be pulled by a matching store ID ($StrID$). The relation contains, e.g., information about the store location, minimum accepted driver's rank, and the bidding preferences (see [9]).

$$\begin{array}{l}\texttt{dlv\_req}(ReqID, Loc_1, Loc_2, Time, MinRank, Pref) \leftarrow \\ \quad \texttt{req}(ReqID, Loc_1, Time, StrID) \text{ WHERE } \texttt{store\_info}(StrID, Loc_2, MinRank, Pref).\end{array}$$

In the above rule pattern, relation $store\_info$ contains *explicit* data. With no restriction, it could also contain a changing (updatable) data; or *implicit* knowledge derived by rules, possibly spanning over multiple relations, involving *recursions* and so forth.

Easy of programming. SQL-based syntax is predominant in today's CEP systems [1,6,14]. It is considered to be easy to understand, as many programmers today are

[4] Complete running implementation of the use case is published under name *ETALIS* on the Event Processing Technical Society web site:
`http://www.ep-ts.com/content/view/79/`

familiar with database concepts. We propose a *rule-based syntax* and argue that it is convenient for CEP. We base our opinion gained on experience in implementation of Fast Flower Delivery use case, as well as, on the implementation of our language. For example, let us consider the following simple pattern rules.

$\texttt{ce1}(Result) \leftarrow \texttt{e}(Name, Result)$ SEQ $\texttt{e}(Name, Result)$ WHERE $(Name =' a', Result = 1)$.
$\texttt{ce2}(Result) \leftarrow \texttt{ce1}(Result)$ AND $\texttt{ce1}(Result)$ WHERE $(Result = 1)$.

Their representation in an SQL-like language of Esper[5] based on [6] is shown below. As we see, complex events detect by the first pattern need to be *re-inserted* in a temporal stream of events $tmpE$ (using $insert$ in the first Esper statement). If complex event $ce2$ was further used in building a more complex event, we would to $insert$ instances of $ce2$ event in another temporal stream too. Consequently, very complex (nested) events in such a language can become easily unreadable. On the other hand, with a rule-based syntax it is easy to nest (complex) events. Also it is easy to pass data within events via *variable binding* which in total gives a more compact and clear syntax of the language.

```
<Query name= "ce1" text="
    insert into tmpE(ceName, Result)
    select 'ce1' as ceName, e1.Result as Result
    from pattern [every ( +
        e1=e(e1.Name='a' and e1.Result=1) ->
        e2=e(e2.Name='a' and e2.Result=1) )]"/>

<Query name= "ce2" text="
    select 'ce2' as Name, e1.Result as Result
    from pattern [every ( +
        e1=tmpE(e1.ceName='ce1' and e1.Result=1) and
        e2=tmpE(e2.ceName='ce1' and e2.Result=1) )]"/>
```

Also it is worth mentioning that our prototype implementation consist of about 2500 lines of Prolog code (see Section 5), while Esper 3.3.0 has approx. 150000 lines of code. Hence rule-based declarative programming results in drastic *reduction* in code size.

Knowledge-based complex event processing. Events and event pattern rules represent *temporal* knowledge, based on which it is possible to derivate more complex dynamic matters. Apart from this knowledge, there may exists *static* (or evolving) knowledge (i.e., facts, rules and ontologies, constituting the *domain* knowledge). We have already seen how static data can be used for event enrichment. More importantly, to detect complex events we can also consult additional (*contextual*) knowledge, e.g., to prove *semantic* relations among matched events (not only temporal relations). While detecting complex events incrementally (at run time), our formalism may additionally evaluate the static knowledge (expressed as Prolog rules and facts) to enhance the detection. In this section we give an example where combining events with static knowledge may be beneficial in practice.

The following rule detects a $route$ event, i.e., a sequence of a delivery assignment event (dlv_assgn) followed by a driver's location event ($gpsToRegion$).

$\texttt{route}() \leftarrow \texttt{dlv\_assgn}(DrvID, Loc_1)$ SEQ $\texttt{gpsToRegion}(DrvID, Loc_2)$ WHERE $\texttt{reachable}(Loc_1, Loc_2)$.

[5] Esper is a CEP system: http://esper.codehaus.org/

To check whether the route is possible, rules defining *reachability* between two location are evaluated.

$\texttt{reachable}(X,Y) \leftarrow \texttt{linked}(X,Y).$
$\texttt{reachable}(X,Z) \leftarrow \texttt{linked}(X,Y), \texttt{reachable}(Y,Z).$

Information about current connections (w.r.t traffic conditions, roads closed due to maintenance etc.) are encoded through the following links.

$\texttt{linked}(L_1, L_2)$
$\ldots$
$\texttt{linked}(L_{n-1}, L_n)$

We see that in order to detect event $route$, an occurrence of event $gpsToRegion$ needs to follow an occurrence of dlv_assgn. Additionally, the pattern will be matched only if the two events carry *reachable* locations (i.e., there exist a *semantic* relation, approved through the *background* knowledge).

Another example demonstrates use of rules for event *translation* and *classification*. Periodically sent drivers' gps events are translated to city region events (for convenience of store dispatchers).

$\texttt{gpsToRegion}(DrvID, Rg) \leftarrow \texttt{gps}(DrvID, coord(SNH, Lat, EWH, Long))$
$\quad \text{WHERE } trsf\_rule(coord(SNH, Lat, EWH, Long), Rg).$

Prolog rules are used as background knowledge to classify gps coordinates into city regions.

$\texttt{trsf\_rule}(\texttt{coord}('N', X, 'W', Y), 'Manhattan') : -4042 < X, X < 4049, 7358 < Y, Y < 7370, !.$
$\ldots$
$\texttt{trsf\_rule}(\texttt{coord}('N', X, 'W', Y), 'StatenIsland') : -4034 < X, X < 4040, 7368 < Y, Y < 7399, !.$

In this section we arguable demonstrated powerful features of our formalism that go beyond (non-logic-based) state-of-the-art CEP systems [1,6,14]. In the following, we explain how complex events are detected using event-driven incremental computation, a feature that is the main difference of our work in comparison to other (logic-based) state-of-the-art CEP approaches [13,16,15,7,17].

4 Run-Time Detection of Complex Events

This section describes how complex events, defined in Section 2.2, are computed at run-time. We explain the pattern matching procedure for a *sequence* of events. In principle the mechanism is similar for other operators too, which we omit due to space restrictions. For details about all operators, an interested reader is referred to [5].

Let us consider a sequence of events represented by rule (1) (e is detected when an event a[6] is followed by b, and followed by c). We can always represent the pattern (1) as $e \leftarrow ((a \text{ SEQ } b) \text{ SEQ } c)$. In general, rules (2) represent two equivalent rules.[7] We refer to this kind of "events coupling" as *binarization* of events. Effectively, in binarization we

[6] More precisely, by "an event a" is meant an *instance* of the event a.
[7] If no parentheses are given, we assume all operators to be left-associative. While in some cases (e.g., SEQ) this is irrelevant, other operators such as PAR are not associative.

introduce *two-input* intermediate events (goals). For example, now we can rewrite rule (1) as $ie_1 \leftarrow a$ SEQ b, and the $e \leftarrow ie_1$ SEQ c. Every monitored event (either atomic or complex), including intermediate events, will be assigned with one or more *logic rules*, fired whenever that event occurs. Using the binarization, it is more convenient to construct *event-driven* rules for three reasons. First, it is easier to implement an event operator when events are considered on "two by two" basis. Second, the binarization increases the possibility for *sharing* among events and intermediate events, when the granularity of intermediate patterns is reduced. Third, the binarization eases the *management* of rules. Each new use of an event (in a pattern) amounts to appending one or more rules to the existing rule set. However what is important for the management of rules, we don't need to *modify* existing rules when adding new ones (even when patterns with negations are added).

$$e \leftarrow a \text{ SEQ } b \text{ SEQ } c. \tag{1}$$

$$\begin{aligned} &e \leftarrow p_1 \text{ BIN } p_2 \text{ BIN } p_3 \ldots \text{ BIN } p_n. \\ &e \leftarrow (((p_1 \text{ BIN } p_2) \text{ BIN } p_3) \ldots \text{ BIN } p_n). \end{aligned} \tag{2}$$

In the following, we give more details about assigning rules to each monitored event. We also sketch an algorithm (using Prolog syntax) for detecting a sequence of events.

Algorithm 4.1 accepts as input a rule referring to a binary sequence $e_i \leftarrow a$ SEQ b, and produces event-driven backward chaining rules (i.e. executable rules) for the sequence pattern. The binarization step must precede the rule transformation. Rules, produced by Algorithm 4.1, belong to one of two different classes of rules. We refer to the first class as to *goal inserting rules*. The second class corresponds to *checking rules*. For example, rule (4) belonging to the first class inserts $goal(b(\_,\_), a(T_1, T_2), e1(\_,\_))$. The rule will fire when a occurs, and the meaning of the goal it inserts is as follows: "an event a has occurred at $[T_1, T_2]$,[8] and we are waiting for b to happen in order to detect ie_1". Obviously, the goal does not carry information about times for b and ie_1, as we don't know when they will occur. In general, the *second* event in a goal always denotes the event that has just occurred. The role of the *first* event is to specify what we are waiting for to detect an event that is on the *third* position.

Rule (5) belongs to the second class being a *checking rule*. It checks whether certain prerequisite goals already exist in the database, in which case it triggers the more complex event. For example, rule (5) will fire whenever b occurs. The rule checks whether $goal(b(T_3, T_4), a(T_1, T_2), ie_1)$ already exists (i.e. a has previously happened), in which case the rule triggers ie_1 by calling $ie_1(T_1, T_4)$. The time occurrence of ie_1 (i.e. T_1, T_4) is defined based on the occurrence of constituting events (i.e. $a(T_1, T_2)$, and $b(T_3, T_4)$, see Section 2.2). Calling $ie_1(T_1, T_4)$, this event is effectively propagated either upward (if it is an intermediate event) or triggered as a finished complex event.

We see that our *backward* chaining rules compute goals in a *forward* chaining manner. The goals are crucial for computation of complex events. They show the current state of progress toward matching an event pattern. Moreover, they allow for determining

[8] Apart from the timestamp, an event may carry other data parameters. They are omitted here for the sake of readability.

Algorithm 4.1. Sequence.

```
Input: event binary goal $ie_1 \leftarrow a$ SEQ $b$.
Output: event-driven backward chaining rules for SEQ operator.
Each event binary goal $ie_1 \leftarrow a$ SEQ $b$. is converted into: {
  $a(T_1, T_2) : -for\_each(a, 1, [T_1, T_2]).$
  $a(1, T_1, T_2) : -assert(goal(b(\_,\_), a(T_1, T_2), e1(\_,\_))).$
  $b(T_3, T_4) : -for\_each(b, 1, [T_3, T_4]).$
  $b(1, T_3, T_4) : -goal(b(T_3, T_4), a(T_1, T_2), ie_1), T_2 < T_3,$
        $retract(goal(b(T_3, T_4), a(T_1, T_2), ie_1(\_,\_))), ie_1(T_1, T_4).$
}
```

the "completion state" of any complex event, at any time. For instance, we can query the current state and get information how much of a certain pattern is currently fulfilled (e.g. notify me if an event is about to happen; for example it is 90% completed). Further, goals can enable *reasoning* over events (e.g. answering which event occurred before some other event, although we do not know a priori what are explicit relationships between these two; correlating complex events to each other; establishing more complex constraints between them etc.).

Goals can persist over a period of time. It is worth noting that *checking rules* can also delete goals. Once a goal is "consumed", it is removed from the database[9]. In this way, goals are kept persistent as long as (but not longer) than needed.

$$\begin{aligned}&for\_each(Pred, N, L) : -((FullPred = ..[Pred, N, L]),\\&event\_trigger(FullPred), (N1 is N + 1),\\&for\_each(Pred, N1, L)) \vee true.\end{aligned} \tag{3}$$

$$a(1, T_1, T_2) : -assert(goal(b(\_,\_), a(T_1, T_2), e1(\_,\_))). \tag{4}$$

$$\begin{aligned}&b(1, T_3, T_4) : -goal(b(T_3, T_4), a(T_1, T_2), ie_1), T_2 < T_3,\\&retract(goal(b(T_3, T_4), a(T_1, T_2), ie_1(\_,\_))), ie_1(T_1, T_4).\end{aligned} \tag{5}$$

Finally, we see that for each different event type (i.e. a and b in our case) we have one rule with a for_each predicate. It is defined by rule (3). Effectively, it implements a loop, which for any occurrence of an event goes through each rule specified for that event (predicate) and fires it. For example, when a occurs, the first rule in the set of rules from Algorithm 4.1 will fire. This first rule will then loop, invoking all other rules specified for a (those having a in the rule head). In our case, there is only one such a rule, namely rule (4). In general case, there may be as many of these rules as usages of a particular event may be manifold in an event program (i.e. set of all event patterns).

Memory management. It is worth mentioning that we have implemented two memory management techniques to *prune* outdated events, and hence free up memory in our running implementation (see Section 5). The first technique modifies the binarization

[9] Removing "consumed" goals is often needed for space reasons but might be omitted if events are required in a log for further processing or analyzing.

step by pushing the time constraints (set by pattern's time window information; users are always encouraged to write patterns with certain time constraints). The technique ensures that time window constraints are checked during the incremental event detection. Therefore unnecessary intermediary sub-complex events will not be generated if the time constraints are violated (i.e., time expired). Our second solution for garbage collection is to prune expired events (goals) by using periodic events, generated by the system. Similarly to the first technique, rules are defined with time window constraints and the binarization pushes the constraints to sub-components. This technique however does not check the constraints at each step during the incremental event detection. Instead, events (goals) are pruned periodically as system events are triggered[10].

5 Implementation and Experimental Results

As a proof of concept, we have provided a prototype implementation of the language. In this section, we present experimental results of our prototype in comparison to Esper 3.3.0[11], i.e., we compare a declarative implementation (written in Prolog) versus a procedural one (written in Java). Esper is an engine primarily relying on state machines, i.e. a different paradigm that is widely used in CEP systems.

The test cases presented here were carried out on a workstation with Intel Core Quad CPU Q9400 2,66GHz, 8GB of RAM, running Windows Vista x64. Since our prototype automatically compiles the user-defined complex event descriptions into Prolog rules, we used SWI Prolog version 5.6.64[12] and YAP Prolog version 5.1.3[13]. All tested engines ran in a single dedicated CPU core.

To run tests, we have implemented an event stream generator, which creates time series data with probabilistic values. Event streams are generated so that every event in a stream is used for detection of one or more complex events (except the test defined by rule (7)). The whole output generated from all tests is validated, so we have made sure that all tested systems produce the same, correct, results.

Figure 3 shows experimental results we obtained for the *sequence* operator (SEQ). In particular, Figure 3(a) shows the throughput measurements for a pattern that exhibits a sequence of three events and the join operation on their Id attribute, see rule (1). The X-axis shows the event throughput achieved by the three different CEP systems: Esper 3.3.0, and our prototype (P) running on SWI and YAP Prolog, denoted as P-SWI and P-YAP respectively). The Y-axis shows different sizes of event streams, used for detection of complex events, defined by rule (6). In this test, our system clearly outperforms Esper. The throughput achieved by YAP engine is more than twice as big as the one produced by Esper. In Figure 3(b) we have evaluated the sequence which (apart from the join operation) also contain a selection parameter K (see rule (7)). K varies selectivity of Y attribute, ranging from 10% till 100%. When 10%-50% of the input events are selected, Esper shows significant advantage over our system. Hence in the future we need to

[10] Frequency of system events can be programmatically scheduled, depending on available system memory.

[11] Esper: `http://esper.codehaus.org`

[12] SWI Prolog `http://www.swi-prolog.org/`

[13] YAP Prolog: `http://www.dcc.fc.up.pt/~vsc/Yap/`.

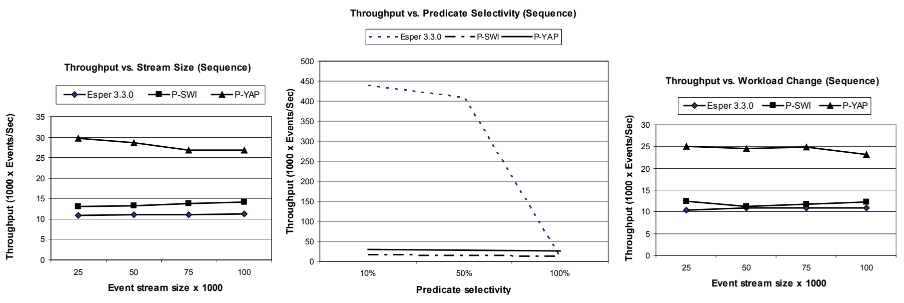


Fig. 3. Sequence - (a) Throughput (b) Throughput vs. Predicate Selectivity (c) Throughput vs. Workload Change

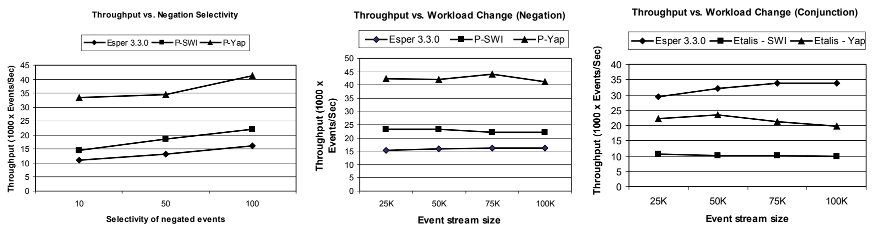


Fig. 4. Negation - (a) Throughput vs. Selectivity (b) Throughput vs. Workload Change (c) Conjunction - Throughput

review our implementation so to select events as early as possible. When all events are taken into account (100% selectivity), our system running on YAP is slightly better than Esper. We did this test on a stream of 25K events. In Figure 3(c) we extended the tests (for 100%) to check out whether the system throughput will remain constant for bigger streams (e.g., 50K-100K).

$$d(Id, X, Y, Z) : -a(Id, X) \text{ SEQ } b(Id, Y) \text{ SEQ } c(Id, Z). \quad (6)$$

Figure 4 presents experimental results for *negation* (NOT) and *conjunction* (AND). Figure 4(a) shows results obtained by evaluating a negated pattern from rule (8). The pattern is detected when an instance of a is followed by an occurrence of b with no c in between the two events. We have generated input event streams with different percentage of occurrences of events of type c (i.e., 10%-100%). We see that our prototype (either run by SWI or YAP) dominates over Esper. The test is computed on a stream of 25K. Figure 4(b) shows that the throughput does not go down even though we increased the stream size (e.g., 50K-100K). We have tested conjunction operator too. The pattern is specified by rule (9), and results are presented in Figure 4(c). Esper was faster in this test. Our algorithm for handling conjunction (see [5]) contains twice as many rules as the algorithm for sequence (i.e., two events in a conjunct may occur in any order). As a future work, we will try to improve the implementation of conjunction by corresponding rules in that algorithm.

$$c(Id, X, Y) : -a(Id, X) \text{ SEQ } b(Id, Y) \text{ WHERE } (Y < K). \tag{7}$$

$$d(Id, X, Y) : -a(Id, X) \text{ SEQ } b(Id, Y) \text{NOT} c(Id, Z). \tag{8}$$

$$d(Id, X, Y) : -a(Id, X) \text{ AND } b(Id, Y) \text{ AND } c(Id, Z). \tag{9}$$

$$d(Id, X, Y) : -a(Id, X) \text{ SEQ } (b(Id, Y) \text{ OR } c(Id, Y)). \tag{10}$$

$$\begin{aligned} tc(X, Y) &: -a(X, Y). \\ tc(X, Y) &: -tc(X, Z) \text{ SEQ } a(Z, Y). \end{aligned} \tag{11}$$

Figure 5(a) shows results for *disjunction*, and evaluation of rule (10). In this test our system running on YAP was the most effective. The throughput for this test is similar to results for sequence (Figure 3(a)); this means that the presence of a disjunct does not ruin the performance of the sequence. We have also tested computation of the transitive closure (see rule (11)). The throughput change for different sizes of event streams are presented in Figure 5(b). Evaluation results were obtained under chronological consumption policy, see [5]. Our system on YAP was the fastest, however the difference between evaluations running on YAP and SWI was huge. Finally, Figure 5(c) compares the tested systems w.r.t event plan sharing. We have run an event program containing the same pattern (similar to rule (6)) multiplying the pattern 1, 8, and 16 times. The focus was on examining how well the systems can exhibit computation sharing among patterns. Our system run by YAP was not resistant on increase of pattern rules. However our prototype executed on SWI was still faster than Esper, see Figure 5(c).

At the end, let us mention that the cost of compilation of an event program (written in the proposed language) into Prolog rules is minor (typically few micro seconds).

In this section, we have provided measurement results of our running CEP engine. Even though there is a lot of room for improvements, preliminary results show that logic-based event processing has the capability to achieve significant performance. Working 15 months on this project, we have managed to develop a CEP language and a corresponding system that is competitive to a mature CEP engines such as Esper 3.3.0. Taking inference capability into account, logic-based CEP goes beyond the state-of-the art providing a powerful combination of *deductive* capabilities and *temporal* features, while at the same time exhibiting competitive run-time characteristics.

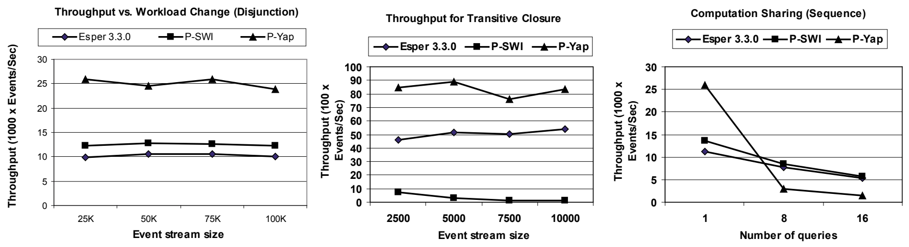


Fig. 5. (a) Disjunction-Throughput (b) Transitive Closure (c) Plan Sharing

6 Conclusions and Future Work

We propose a language for Complex Event Processing based on deductive rules. The language comes with a clear declarative, formal semantics for complex event patterns. We have also provided a prototype implementation of the language, which allows for specification of complex events and their detection at occurrence time. Our approach clearly substantiates existing event-driven systems with declarative semantics and the power of knowledge-based event processing.

Acknowledgments

This work was supported by European Commission funded project SYNERGY (FP7-216089). We thank Jia Ding and Ahmed Khalil Hafsi for their help in the implementation and testing of ETALIS prototype.

References

1. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: SIGMOD (2008)
2. Alferes, J.J., Banti, F., Brogi, A.: An event-condition-action logic programming language. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 29–42. Springer, Heidelberg (2006)
3. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
4. Alves, A.: Extensions to logic programming inference engines to support cep. In: RuleML '09 (2009)
5. Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., Studer, R.: Etalis: Rule-based reasoning in event processing. In: Helmer, S., Poulovassilis, A., Xhafa, F. (eds.) Reasoning in Event-based Distributed Systems, Studies in Computational Intelligence Series. LNCS. Springer, Heidelberg (2010)
6. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: Semantic foundations and query execution. In: VLDB Journal (2003)
7. Bry, F., Eckert, M.: Rule-based composite event queries: The language xchangeeq and its semantics. In: Marchiori, M., Pan, J.Z., Marie, C.d.S. (eds.) RR 2007. LNCS, vol. 4524, pp. 16–30. Springer, Heidelberg (2007)
8. Chakravarthy, S., Mishra, D.: Snoop: an expressive event specification language for active databases. In: Data Knowledge Engineering, Elsevier Science Publishers B. V., Amsterdam (1994)
9. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Co., Greenwich (2010)
10. Forgy, C.L.: Rete: A fast algorithm for the many pattern/ many object pattern match problem. Artificial Intelligence (1982)
11. Gatziu, S., Dittrich, K.R.: Samos: an active object-oriented database system. In: IEEE Bulletin of the TC on Data Engineering (1992)
12. Haley, P.: Data-driven backward chaining. In: International Joint Conferences on Artificial Intelligence, Milan, Italy (1987)
13. Kowalski, R., Sergot, M.: A logic-based calculus of events. In: New Generation Computing, Ohmsha (1986)

14. Krämer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. ACM Trans. Database Syst. (2009)
15. Lausen, G., Ludäscher, B., May, W.: On active deductive databases: The statelog approach. In: ILPS’97 (1998)
16. Miller, R., Shanahan, M.: The event calculus in classical logic - alternative axiomatisations. Electron. Trans. Artif. Intell. (1999)
17. Paschke, A., Kozlenkov, A., Boley, H.: A homogenous reaction rules language for complex event processing. In: International Workshop on Event Drive Architecture for Complex Event Process. ACM, New York (2007)

Analyzing the AIR Language: A Semantic Web (Production) Rule Language*

Ankesh Khandelwal[1], Jie Bao[1], Lalana Kagal[2], Ian Jacobi[2], Li Ding[1], and James Hendler[1]

[1] Rensselaer Polytechnic Institute
Troy, NY 12180
{ankesh,baojie,dingl,hendler}@cs.rpi.edu
[2] Massachusetts Institute of Technology
Cambridge, MA 02139
{lkagal,jacobi}@csail.mit.edu

Abstract. The **A**ccountability **I**n **R**DF (*AIR*) language is an N3-based, Semantic Web production rule language that supports nested activation of rules, negation, closed world reasoning, scoped contextualized reasoning, and explanation of inferred facts. Each AIR rule has unique identifier (typically an HTTP URI) that supports reuse of rule. In this paper we analyze the semantics of AIR language by: i) giving the declarative semantics that support the reasoning algorithm, ii) providing complexity of AIR inference; and iii) evaluating the expressiveness of language by encoding Logic Programs of different expressivities in AIR.

1 Introduction

AIR was developed as an extension to N3Logic [4] to support accountable privacy protection in Web-based information systems [11]. While N3Logic supports scoped contextualized reasoning (SCR), nested graphs, and built-in functions, AIR additionally supports rule reuse, rule nesting and explanation of rule-based inference. AIR rules are encoded in N3[1], which extends the expressiveness of RDF with (i) the ability to use graphs as literal values, (ii) universal or existential quantification of variables in a graph and (iii) built-in functions and operators represented as RDF properties.

AIR is designed to provide detailed explanations for actions performed by AIR reasoner. AIR permits natural language descriptions to be added to actions, so that when an action is performed the description is included in the justification, by the reasoner. Since justifications can potentially reveal sensitive information from the knowledge base or expose bias in the rules, AIR also provides ways to customize explanations, e.g. hiding actions of certain rules.

* This work was supported in part by NSF Cybertrust award CNS-0831442, IARPA award FA8750-07-2-0031, AFOSR award FA9550-09-1-0152, and support to RPI from the Information Technology Alliance in Network and Information Science by U.S. ARL and the U.K. MoD under Agreement Number W911NF-06-3-0001.

[1] http://www.w3.org/DesignIssues/Notation3.html

AIR rules have unique IDs (IRIs), which enables the rules to be part of the linked data cloud and reused by other rule definitions. Furthermore, AIR rules can be nested. These features are natural to model real-world rules and laws where conditions are often split into sequentially-activated rules, and rules are frequently reused.

AIR supports rule definitions in which Web resources, such as triple stores with SPARQL [18] end-points, can be objectively checked for patterns. Sometimes the data has an associated intensional component, defined through rules. For complete access to the data, AIR supports inclusion of certain inferences without allowing the rules in the intensional component to be applied to the entire input data. SCR is important for a Web rule languages because information on the Web is often assumed to be incomplete, and its correctness is subject to the trustworthiness of the source.

There are many rule languages and rule systems; some are more advanced while others are at prototypical levels. Liang et. al. give a nice overview of popular, and often advanced, rule systems in [15]. Examples of rule languages include N3Logic, NG [20], SILK [8], and SWRL [10], while some rule engines include Jess[2], Jena[3], and XSB[4]. These languages and systems have different levels of expressiveness and features.

Networked Graphs (NGs) supports the well-founded (WF) negation, XSB supports Negation as Failure (NAF), and Jess, a production rule system, supports non-logical negation. SILK supports WF negation, as well as classical negation. SWRL and Jena do not support negation. In comparison, N3Logic supports monotonic negation, and AIR supports non-monotonic negation, which is different from negation accepted by the systems above.

Rule Interchange Format (RIF) [14] has two major dialects – the Framework for Logic Dialects (FLD) and the Production Rule Dialect (PRD). Unlike PRD, actions in AIR cannot modify or remove facts, but they can add new production rules. FLD is an extensible framework for rule-based languages, and includes the Basic Logic Dialect (BLD), which corresponds to the definite Horn rules with equality and standard first-order logic semantics. AIR is neither more nor less expressive than BLD. Unlike BLD, function symbols are not supported in AIR, whereas AIR supports negation. SWRL is a function-free rule language limited to binary and unary predicates, and all its features, barring different-from, are covered by BLD. AIR is as expressive as SWRL, and also supports negation.

OPS/YES [21] extends the OPS5 [7] production rule system with many features including *incremental rule addition*, which allows matching in the actions. AIR has a similar notion of rule nesting. Other than AIR, of the systems mentioned above, only Jena supports (partially) nesting of rules.

In SPARQL, query patterns can be restricted to selective named graphs, and as a result NGs naturally supports SCR. Other than N3Logic, AIR, and NGs, SILK also supports SCR.

[2] http://www.jessrules.com/

[3] http://jena.sourceforge.net/

[4] http://xsb.sourceforge.net/

The Ontonova system [1] provides natural language explanation of proof trees for conclusions by the Ontobroker[5] through meta-inferencing. A similar mechanism for generating proofs for defeasible reasoning system DR-DEVICE is described in [3]. In contrast, the explanations for conclusions in AIR are generated by the reasoner itself, and the justification can be declaratively modified by changing rule definitions.

AIR's focus on explanation generation for Web reasoning makes it unique. Other distinguishing features include AIR's ability to treat AIR rules as part of linked data and the ability to match patterns against remote triple stores.

While AIR has a production rule syntax, it is limited to assertions of facts, and addition of rules. Neither can facts be removed from the current state of the world nor any procedural attachments may be called. Therefore, we can define its declarative semantics. In this work we define the model-theoretic semantics of AIR-programs, by defining the translation of an arbitrary AIR-program to a semantically equivalent stratified Logic Program (LP).

There has been an earlier work on translating Jess to and from Situated Ordinary LP (SOLP)[6]and Situated Courteous LP (SCLP) [9], showing interoperability of the two rule languages. However, Jess and AIR are very different rule languages. Jess, unlike AIR, supports procedural attachments and with certain restrictions on its features has been shown to support (stratified) NAF [9]. Not only is the notion of nesting of rules absent from Jess, AIR's negation is different from NAF. The authors are not aware of any other related works.

The remainder of the paper is organized as follows. Section 2 provides an overview of the AIR language and reasoning. In section 3, we define a new class of stratified LP - Positively Stratified Negatively Hierarchical LP (PSNHLP). In section 4, we define the declarative semantics, and provide the data and program complexities of AIR programs. In section 5, we show how LP rules and fairly expressive LPs, such as PSNHLP, can be encoded in AIR. We conclude in section 6 with a summary and discussion on future work.

There is a longer version of the paper [13][7], that includes review of standard terminologies used in Logic Programming, proofs of all the claims, and detailed comparison of AIR with other rule systems.

2 AIR Overview

The AIR production rule system has two components- the rule language and the rule engine (reasoner). The reasoner computes the closure, **AIR-closure**, for given facts, in N3, with respect to given AIR-program. An **AIR-program** contains one or more AIR-rules. The closure contains the initial facts and all the facts that can be deduced from it using the given rules. We will review the AIR language, and introduce the algorithm for computing the closure in sections 2.1 and 2.2. The details about AIR justification are available at [12].

[5] http://ontobroker.semanticweb.org/

[6] http://sweetrules.projects.semwebcentral.org/sweetrules-overview-presentation-2005-04-24-v3.pdf

[7] http://tw.rpi.edu/proj/tami/AIR_Language_Formalization

A. RULE SET
# $m_1 \geq 0,\ m_2 > 0$
@forAll $< u\_var_1 >, \ldots, < u\_var_{m_1} >$.

<setid> a air:RuleSet ;
    air:rule $< ruleid_1 >, \ldots, < ruleid_{m_2} >$.

B. RULE
# $n_1, n_2, n_3, n_4 \geq 0, n_3 + n_4 > 0$
<ruleid> a air:BeliefRule ; # alternatively air:HiddenRule, air:EllipsedRule
    air:if { @forSome $< e\_var_1 >, \ldots, < e\_var_{n_1} >$.
        $s_1 < p_1 > o_1$.
        ...
        $s_{n_2} < p_{n_2} > o_{n_2}$. } ;
    air:then $< t\_action_1 >, \ldots, < t\_action_{n_3} >$;
    air:else $< e\_action_1 >, \ldots, < e\_action_{n_4} >$.

C. ACTIVATING NEW RULE
<action> air:description (list_of_var_and_str) ;
    air:rule $< ruleid >$.

D. ASSERTING A GRAPH PATTERN
# $n > 0$
<action> air:description (list_of_var_and_str) ;
    air:assert {$s_1 < p_1 > o_1$.
        ...
        $s_n < p_n > o_n$. } .

Fig. 1. Template for defining AIR-program

2.1 AIR Language

AIR rules are defined using following properties: `air:if`, `air:then`, `air:else`, `air:description`, `air:rule` and `air:assert`, according to the AIR abstract syntax[8]. A template for declaring an AIR-program is shown in Figure 1. This template will be used in the later sections. When we refer to s and o related by `p1.p2` we mean that `s p1 [ p2 o ]` holds.

The rules are of the form `air:if` *condition*; `air:then` *then-actions*; `air:else` *else-actions*. The condition is specified as a graph pattern, similar to the Basic Graph Pattern of SPARQL queries, which may be pattern matched against N3 graphs. Whenever the condition matches the current state of the world, i.e. the facts known or inferred so far, variables may acquire bindings and all the `then-action`s are fired. Otherwise all the `else-action`s are fired.

The rules are grouped under rule-sets. The rules nested directly under the `RuleSet` are referred to as the **top rule**s of the rule-set. A rule (**child rule**) nested under another rule (**parent rule**) via `air:then.air:rule` is said to be positively nested and a rule nested via `air:else.air:rule` is said to be negatively nested. A **chain** of rules is defined as a sequence of rules such that every rule barring the first in the chain is nested under the preceding rule. $\texttt{t_action}_i$s and $\texttt{e_action}_i$s, in Figure 1, are the `then-action`s and `else-action`s, respectively. Together they are referred to as `action`s. The actions can be annotated with natural language description through the `air:description` property. Note that the description can be defined using variables.

[8] http://tw.rpi.edu/proj/tami/AIR_Abstract_Syntax

```
@prefix air:<http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
@prefix conf:<http://www.conf.org/ontology#>.
@prefix pol:<http://www.conf.org/policies/publication#>.
@prefix:<http://www.conf.org/policies/publication#>.

pol:PubInProcPolicy a air:RuleSet ;
    air:rule pol:CheckPub .

@forAll :PUBL .
pol:CheckPub a air:BeliefRule ;
    air:if { @forSome :PROC . <http://www.conf.org> conf:hasProceedings :PROC .
             :PROC conf:hasPaper :PUBL . } ;
    air:then [ air:description (:PUBL " published in this conference") ;
               air:rule pol:ChkNonCompl ],
             [air:rule pol:CheckAuth], [air:rule pol:CheckExempt].

pol:ChkNonCompl a air:BeliefRule ;
    air:if { :PUBL air:compliant-with pol:PubInProcPolicy . } ;
    air:else [ air:description ("the publication of " :PUBL " is questionable as
                                it did not meet any of the two criteria") ;
               air:assert { :PUBL air:non-compliant-with pol:PubInProcPolicy } ] .

@forAll :AUTH .
pol:CheckAuth a air:BeliefRule ;
    air:if {:PUBL conf:hasAuthor :AUTH.<http://www.conf.org> conf:registeredBy :AUTH.};
    air:then [ air:description ("Author," :AUTH " registered for the conference");
               air:assert { :PUBL air:compliant-with pol:PubInProcPolicy } ] .

@forAll :REASON .
pol:CheckExempt a air:BeliefRule ;
    air:if { @forSome :COCHAIR, :EXEMPTION .
             :COCHAIR a conf:Co-Chair . :PUBL conf:hasFirstAuthor :AUTH .
             :EXEMPTION a conf:PublicationExemption ; conf:exemptedBy :COCHAIR ;
                   conf:exemptee :AUTH ; conf:reason :REASON . } ;
    air:then [ air:description ("the first author " :AUTH " was exempted by one of the
                                                   cochairs, because " :REASON);
               air:assert { :PUBL air:compliant-with pol:PubInProcPolicy } ] .
```

Fig. 2. Example AIR-program, describing the publication policy

The existentially quantified variables may be declared within the condition. The universally quantified variables are declared outside of the rule. The scope of existentially quantified variable is the condition where it's declared, and of universally quantified variable is any rule chain with the first rule as a top rule.

The graph pattern asserted in the action is declared using `air:assert`, and the nested rule is declared using `air:rule` property. The asserted graph patterns cannot contain blank nodes or existentially quantified variables. When the nested rule is activated, an instance of the rule with known variable bindings substituted is created. When a rule with an `air:else` property is activated, its condition cannot contain unbounded universally quantified variables.

Traditionally, rule languages have been designed assuming that all rules will apply on all input data. However, this is not desirable for a web rule language [4,17], and therefore AIR supports SCR. The matching of a condition pattern can be scoped to web documents, RDF-stores with SPARQL end-points, or to the AIR-closure of some facts and AIR-programs using the `log:semantics.log:includes`, `sparql:queryEndPoints.log:semantics.log:includes` and `air:justifies`

A. Data Structures

```
rule = (ruleid, condition, success, then-actions, else-actions)
then(else)-actions = list of child rules to be activated and graphs to be asserted
RB = set of active rules
FB = set of facts
SRF = queue of (rule, bindings) tuples
FRF = queue of rules
```

B. COMP-CLOSURE

```
INPUT : AIR-program, facts
OUTPUT : Closure of facts w.r.t. the AIR-program in FB
1. Initialize the data-structures
  1.1. FB = facts
  1.2. RB = $\phi$
  1.3. SRF = empty queue
  1.4. FRF = empty queue
  1.5. FRF' = empty queue
2. For each top rule, of all RuleSets in AIR-program
  2.1. add-to-rulebase(R)
3. while SRF and FRF is non empty
  3.1. while SRF is non empty
    3.1.1. f = SRF.dequeue(), i.e. remove first queue element and assign it to f
    3.1.2. rule-fire(f.rule.then-actions, f.bindings)
  3.2. while FRF is non empty
    3.2.1. f = FRF.dequeue()
    3.2.2. if f.rule.success is false
      3.2.2.1. FRF'.enqueue(f)
  3.3. while FRF' is non empty
    3.3.1. f = FRF'.dequeue()
    3.3.2. rule-fire(f.rule.else-action, {})
```

C. RULE-FIRE

```
INPUT : actions, bindings bndgs
OUTPUT : updated RB and FB
1. for action in actions :
  1.1. substitute all variables in action which have bindings with bound values from bndgs
  1.2. if action is a rule
    1.2.1. add-to-rulebase(action)
  1.3. if action is a graph
    1.3.1. add-to-factbase(action)
```

D. ADD-TO-RULEBASE

```
INPUT : rule
OUTPUT : updated RB and (SRF or FRF)
1. if rule not in RB
  1.1. add rule to RB
  1.2. R.success = False
  1.3. pattern match rule.condition against FB
  1.4. If rule.condition matched a sub-graph
    1.4.1. R.success = True
    1.4.2. for each sub-graph that matched rule.condition
      1.4.2.1. SRF.enqueue((rule, bindings))
  1.5. If rule.condition did not match any sub-graph
    1.5.1. FRF.enqueue(rule)
```

E. ADD-TO-FACTBASE

```
INPUT : graph (grounded graph pattern)
OUTPUT : updated FB, RB and SRF
1. let graph' = graph $\setminus$ FB
2. if $graph' \neq \phi$
  2.1. add triples in graph' to FB
  2.2. for each rule in RB
    2.2.1. pattern match rule.condition against FB
    2.2.2. If rule.condition matched a sub-graph $g$ s.t. that $g \cap$ graph' $\neq \phi$
      2.2.2.1. rule.success = True
      2.2.2.2. for each sub-graph $g$ that matches the rule.condition and $g \cap$ graph' $\neq \phi$
        2.2.2.2.1. SRF.enqueue((rule, bindings))
```

Fig. 3. The **COMP-CLOSURE** algorithm, and related data-structures and algorithms

built-ins respectively. Note that by using `air:justifies`, a subset of rules can be applied to a subset of data, if desired, without risking the application of those rules to the entire input data. In addition to the built-ins for scoped reasoning, AIR supports most of the N3Logic built-ins[9] for cryptographic, math, string, list and time functions.

Figure 2 shows an example AIR-program. It expresses the conference policy: any paper accepted in the conference should be published in the proceedings only if one of the authors has presented it in the conference (`pol:CheckAuth`), or the first author has received an exemption from one of the co-chairs (`pol:CheckExempt`). The former condition is interpreted in practice by checking to see if one of the co-authors has registered for the conference. It is assumed that the conference activities are logged using the same vocabulary as that used in the

[9] http://www.w3.org/2000/10/swap/doc/CwmBuiltins

```
@prefix colog:<http://www.conf.org/log#>. @prefix conf:<http://www.conf.org/ontology#>.
<http://www.conf.org>   conf:hasProceedings   colog:proc .
colog:proc              conf:hasPaper         colog:pub1 .
colog:pub1              conf:hasAuthor        colog:auth1 .
<http://www.conf.org>   conf:registeredBy     colog:auth1 .
```

Fig. 4. Example of input facts: sub-graph from log of conference activities

rules. `pol:CheckPub` is the only top rule, and the three rules, `pol:ChkNonCompl`, `pol:CheckAuth`, and `pol:CheckExempt` are its child rules.

2.2 AIR Reasoning (The Procedural Semantics)

The AIR reasoner employs a RETE [6] based forward-chaining approach to compute the AIR-closure. Figure 3 describes the **COMP-CLOSURE** algorithm used for computing the AIR-closure. The data structures and algorithms described here are abstractions of the actual implementation [12]. For instance **FB** (Fact Base), which is shown to be a set of facts, is actually an efficiently indexed triple store, and **RB** (Rules Base), the set of active rules, is compiled into the RETE framework. The abstraction is sufficient for analyzing the language.

One cycle of step 3 of **COMP-CLOSURE** is referred to as a **stage**, i.e. steps 3.1 to 3.3 (3.3.1 and 3.3.2 included) constitute one stage of the algorithm. Furthermore, step 3.1 is referred to as a positive stage, denoted by $\textbf{stage}^+$ and steps 3.2 and 3.3 constitute a negative stage, denoted by $\textbf{stage}^-$. We will be using stage_i, stage_i^+ and stage_i^- to refer to i^{th} stage, stage^+ and stage^-, respectively.

In any given stage, successful rules are given priority over failed rules, and their `then-actions` are effected in stage^+ before failed rules fire. When all successful rules have fired, the world is temporarily closed and the `else-actions` of all the failed rules are fired in stage^- with the belief that the conditions of all the failed rules are false. AIR reasoning enters the next stage once the failed rules have all fired. The **COMP-CLOSURE** algorithm computes the next stage until the fixpoint is reached, i.e. when both the **SRF** and **FRF** are empty.

Consider the input facts in Figure 4 and the example AIR-program from Figure 2. The AIR-closure contains one additional triple `colog:pub1 air:compliant-with pol:PubInProcPolicy`. In order to compute the closure, **FB** is initialized by triples in the input and `pol:CheckPub` is added to the **RB**. When the rule is added, its condition matches and it is added to the **SRF** with `pol:PUBL` bound to `colog:pub1`. When it fires, the three child rules are added to the **RB**. Only `pol:CheckAuth` succeeds and other rules are added to the **FRF**. The `pol:CheckAuth` is added to the **SRF** with `:AUTH` bound to `colog:auth1`. It fires next, and asserts the triple `colog:publ1 air:compliant-with pol:PubInProcPolicy`. When that triple is added, it satisfies the `pol:ChkNonCompl`'s condition and the rule's success attribute is set to true. **SRF** is now empty and `pol:ChkNonCompl` is removed from **FRF**. `pol:CheckExempt` is then added to **FRF**. However, it doesn't have an `air:else` property. The fixpoint is reached at the end of first stage.

3 Positively Stratified Negatively Hierarchical Logic Programs

We will review definitions of *Ground*(R), the T_P *operator* and the *perfect model* semantics for stratified LPs, before defining PSNHLPs. *Ground*(R) refers to the set of all possible ground instances of R, with respect to given universe of constant symbols U, where R is a rule or an LP. T_P is the one-step induction operator [2] associated with an LP P, defined by :

$$T_P(I) = \{\ A_0 \mid P \text{ contains a rule whose instantiation is } A_0 \leftarrow A_1, \ldots, A_n \text{ such that } \{A_1, \ldots, A_n\} \subseteq I \text{ holds } \}$$

where I is the given set of atoms that are true. T_P^ω is the fix-point operator, i.e. T_P is applied repeatedly until the fix-point is reached.

The semantics of a (locally) stratified LP, P, is defined in terms of its *perfect model*, which we refer to by *PM*(P). P entails a ground atom A iff A belongs to *PM*(P). Let $P_1 \cup \ldots \cup P_n$ be the (local) stratification of P (*Ground*(P)). We can choose a stratification and assignment of levels to predicates such that the rule $A \leftarrow L_1, \ \ldots, \ L_m$ in P is in P$_i$ iff the level assigned to A is i, and we will assume that to be the case in rest of the paper. *PM*$(P) = M_n$ [19], which is defined by :

$$M_1 = T^\omega_{P_1}(\phi),\ M_2 = T^\omega_{P_2}(M_1), \ldots,\ M_n = T^\omega_{P_n}(M_{n-1})$$

When a rule contains a negative literal in the body, we will refer it as **negative rule**. All other rules are thus **positive rules**. An LP P is **PSNHLP** if there is an assignment of ordinal levels to predicates such that whenever a predicate appears in the body (negatively or positively) of a negative rule, the predicate in the head of that rule is of strictly higher level, and whenever a predicate appears in the body of a positive rule, the predicate in the head has at least that level. Similarly, an LP P is locally PSNHLP if such an assignment of ordinal levels is possible over ground atoms for *Ground*(P). Every hierarchical LP is PSNHLP, and every PSNHLP is stratified LP. This doesn't hold in opposite direction.

Let $P_1 \cup \ldots \cup P_n$ be the stratification of a PSNHLP P, and let $P_i^+ \cup P_i^-$ is the partition of P$_i$ such that a rule from P$_i$ is in P$_i^+$ if it is positive and in P_i^- otherwise. Then, the **PSNH-stratification** of P is defined as $P = P_1^+ \cup P_1^- \cup \ldots \cup P_n^+ \cup P_n^-$. The predicates in the body of a rule in P_i^- have levels strictly smaller than i. The *PM*(P) $=$ M$_n$, where M$_n$ is defined by:

$$\begin{aligned}
&M_1 = T^\omega_{P_1}(\phi),\\
&M_2' = T_{P_2^-}(M_1),\ M_2 = T^\omega_{P_2^+}(M_2'),\\
&\ldots\\
&M_n' = T_{P_n^-}(M_{n-1}),\ M_n = T^\omega_{P_n^+}(M_n')
\end{aligned}$$

such that $M_1 = T^\omega_{P_1}(\phi)$, and $M_i = T^\omega_{P_i}(M_{i-1})$ for $i > 1$. The former holds because $P_1 = P_1^+$ (i.e. $P_1^- = \phi$). The latter follows from following claim.

Claim 1: $T^\omega_{P_i}(M_{i-1}) = T^\omega_{P_i^+}(T_{P_i^-}(M_{i-1}))$

4 Declarative Semantics

The declarative semantics of AIR-program can be defined in terms of a semantically equivalent PSNHLP. We will first define τ, that translates AIR-program Δ to LP $\wp$. We will then show that $\wp$ is PSNHLP and that the AIR-closure of Δ and input facts κ, mapped as ground facts in $\wp$, is same as $PM(\wp)$. $\wp$ contains rules for predicates `t`, $\mathtt{t}_s$, `active_rule` and `cond`, with following meanings :

– `t(s, p, o)` represents N3 triple {`s p o`}.
– $\mathtt{t}_s$`(s, p, o, c, n)` represents true N3 triple {`s p o`}, in the `n`th stage of reasoning, in context `c`.
– `active_rule(ruleid, ?v`$_1$`, ..., ?v`$_m$`, c, n)` represents active AIR-rule, `ruleid`, in the `n`th stage, in context `c`. The exact instance of the rule is determined by the partial bindings of `?v`$_i$'s. `?v`$_i$'s are the universally quantified variables used in the chain of rules to which `ruleid` belongs. For example, the rule `pol:CheckAuth` has two variables `?PUBL` and `?AUTH`.
  If, for example, `active_rule(pol:CheckAuth, colog:pub1, ?AUTH, c, 1)` atom is true, then an instance of the rule `pol:CheckAuth` is active at the 1st stage with `?PUBL` bound to `colog:pub1` in the context `c`. Here `?AUTH` is unbound and can be bound by any constant from U.
– `cond(ruleid, ?cv`$_1$`, ..., ?cv`$_m$`, c, n)` represents a satisfied condition of the AIR-rule, `ruleid`, in the `n`th stage, in context `c`. The `?cv`$_i$'s are universally or existentially quantified variables occurring in the rule's condition. The VAR_IN_COND function returns all these variables for given rule. If, for example, `cond(pol:CheckAuth, colog:pub1, colog:auth1, c, 1)` atom is true then the condition of `pol:CheckAuth` rule is true in 1st stage, in context c, when `?PUBL` and `?AUTH` are bound to `colog:pub1` and `colog:auth1` respectively.
– `succ` (successor) and `=` predicates are used with their usual meanings.

The **context** may be defined by a collection of N3 and/or AIR-program sources, and is identified with an ID. We must use unique predicate symbols to distinguish `active_rule` and `cond` predicates for same rule in different rule chains.

$\tau(\Delta)$ is defined by conjunction of the rules returned from `TRANS(setid)` and `TRANS(ruleid)`, for every rule set and production rule in Δ, respectively. Figure 5 gives the definition of **TRANS**`(ele, ?n)` function, for `ele`'s in a built-in-free AIR-program. The `ele`s are described as per the template in Figure 1, and `?n` is optional argument.

The **MAP**`(val)` function maps `val` to `?val` or `val` depending on whether `val` is declared as a variable or not. `UB` is the upper bound on number of stages required for computing the fix-point for given AIR-program and initial facts. `UB` is finite because assertions can use only fixed IRIs (i.e. no blank nodes are allowed in the asserted graph), and every fact and rule instance is asserted at most once. `UB` is polynomial in the size of input facts and the maximum depth of nesting in input AIR-program [13]. `?n` $\leq$ `UB` ensures that the closure for $\wp$ is finite. Additional rules that hold in every $\wp$ are shown in Figure 6.

The triples in κ are translated as facts into $\wp$, in translation step (TS) 8. The triples asserted and the rules activated when the rule condition is satisfied are

```
A. TRANS(setid), where setid is a RULE-SET
1. ruleset(setid, c) .
2. for i in 1 to $m_2$
       active_rule(ruleid$_i$, ?u_var$_1$, . . ., ?u_var$_{m_1}$, ?c, 1) $\leftarrow$ ruleset(setid, ?c) .
B. TRANS(ruleid), where ruleid is a RULE
3. cond(ruleid, VAR_IN_COND(ruleid), ?c, ?n) $\leftarrow$ t$_s$(MAP(s$_1$), MAP(p$_1$), MAP(o$_1$),
           ?c, ?n), . . ., t$_s$(MAP(s$_{n_2}$), MAP(p$_{n_2}$), MAP(o$_{n_2}$), ?c, ?n), ?n $\leq$ UB .
4. for i in 1 to $n_3$
       TRANS(t_action$_i$, ?n) $\leftarrow$ active_rule(ruleid, ?u_var$_1$, . . ., ?u_var$_{m_1}$, ?c, ?n),
                    cond(ruleid, VAR_IN_COND(ruleid), ?c, ?n), ?n $\leq$ UB .
5. for i in 1 to $n_4$
       TRANS(e_action$_i$, ?m) $\leftarrow$ active_rule(ruleid, ?u_var$_1$, . . ., ?u_var$_{m_1}$, ?c, ?n),
            not(cond(ruleid, VAR_IN_COND(ruleid), ?c, ?n)), succ(?m, ?n), ?n$\leq$UB.
C. TRANS(action, ?n), where action is ACTIVATING NEW RULE
6. active_rule(ruleid, ?u_var$_1$, . . ., ?u_var$_{m_1}$, ?c, ?n)

D. TRANS(action, ?n), where action is ASSERTING A GRAPH PATTERN
7. t$_s$(MAP(s$_1$),MAP(p$_1$),MAP(o$_1$), ?c, ?n), . . ., t$_s$(MAP(s$_n$),MAP(p$_n$),MAP(o$_n$),?c,?n)

E. TRANS({s p o}), where {s p o} is INPUT FACT
8. t$_s$(s, p, o, c, 1) .
```

Fig. 5. TRANS(`ele`, `?n`) definition, where `ele` is defined according to the template in Figure 1

true from the same stage (TS-4), whereas they are true only from next stage when the rule condition failed (TS-5). The definition in Figure 5 can be extended to handle the built-ins for SCR, namely `air:justifies`, `log:includes`, and `log:notIncludes` [13].

For example, TRANS(`pol:ChkNonCompl`) yields following rules :

- t_s(?PUBL , `air:non-compliant-with`, `pol:PubInProcPolicy`, ?c, ?m) $\leftarrow$
  active_rule(`pol:ChkNonCompl`, ?PUBL, ?c, ?m), succ(?n, ?m), ?n $\leq$ UB,
  not(cond(`pol:ChkNonCompl`, ?PUBL, ?c, ?n)) .
- cond(`pol:ChkNonCompl`, ?PUBL, ?c, ?n) $\leftarrow$
  t_s(?PUBL,`air:compliant-with`, `pol:PubInProcPolicy`, ?c, ?n) .

Claim 2: $\wp$ is locally PSNHLP.
Proof. Let the assignment of levels to ground atoms for all predicates except `t`, `succ`, and = be equal to the value of their last term. For example $\mathtt{t}_s$`(s, p, o, c, n)` is assigned level `n`. Let predicates `t`, `succ`, and `=` be assigned levels `UB`, 1, and 1, respectively. This assignment gives a local PSNH-stratification of $\wp$. □

We can replace the atom $\mathtt{t}_s$`(s, p, o, c, n)` by $\mathtt{t}_{s_n}$`(s, p, o, c)`. We can do so for other atoms with $\mathtt{t}_s$, `active_rule` and `cond` predicate symbols, by introducing $\mathtt{t}_{s_n}$, `active_rule`$_n$ and `cond`$_n$ predicate symbols, for all $n \leq$ `UB`. Although the TS-9, TS-10 and TS-11 rules will expand O(`UB`^{2}) times and the rest O(`UB`) times, changes to TSs in Figures 5 and 6 are straightforward. Then the resulting $\wp$, say $\wp'$, is a PSNHLP as against being just locally PSNHLP.

9. active_rule(?x$_1$,. . .,?x$_m$, ?c, ?n)←active_rule(?x$_1$,. . .,?x$_m$, ?c,?m), ?m≤?n, ?n≤UB.
10. t$_s$(?s, ?p, ?o, ?c, ?n) ← t$_s$(?s, ?p, ?o, ?c, ?m), ?m ≤ ?n, ?n ≤ UB .
11. t(?s, ?p, ?o) ← t$_s$(?s, ?p, ?o, c, ?n), ?n ≤ UB .

Fig. 6. Additional rules that hold in every $\wp$

Let, $Ground(\wp) = \wp_1^+ \cup \wp_1^- \cup \ldots \cup \wp_{\mathrm{UB}}^+ \cup \wp_{\mathrm{UB}}^-$, be the local PSNH-stratification of $\wp$. We can show semantic equivalence, denoted by $\sim$, between the steps for computation of $PM(\wp)$ and the AIR-closure of Δ and κ. $T \sim s$ is used to denote the direct correspondence between extensions of predicates `active_rule` and $\mathtt{t}_s$, inferred after the application of T operator during the computation of $PM(\wp)$, and the rules activated and triples asserted after completion of stage s of **COMP-CLOSURE** execution.

Claim 3: $T^{\omega}_{\wp_1^+} \sim stage_1^+$. For $i > 1$, $T_{\wp_i^-} \sim stage_{i-1}^-$ *and* $T^{\omega}_{\wp_i^+} \sim stage_i^+$.
Let $\aleph$ be the N3 graph obtained by taking the N3 representation of extensions of the predicate `t` in $PM(\wp)$.

Claim 4: The AIR-closure of Δ and κ is same as $\aleph$.

Proof. From claim 3 it follows that for all i ≥ 1, $M_i \cap \aleph$ and the triples in **FB** after the completion of $stage_i^+$ are the same. Since the fix-point is reached by the end of UB$^{\mathrm{th}}$ stage, $M_{UB} \cap \aleph$ and the triples in the AIR-closure of Δ and κ are the same. However, $PM(\wp) = M_{UB}$. □

The above results (claims 3 and 4) hold for $\wp'$ as well [13]. We have seen that $\wp$ can be viewed as a stratified LP (PSNHLP), and we have shown above the direct correspondence between the steps for computing $PM(\wp)$ (equivalently $PM(\wp')$) and those for computing AIR-closure of Δ and κ. Therefore, complexity results for AIR follow from those for stratified programs [5]. The **data complexity** is the complexity of computing the model when the rules-base is fixed and the fact-base is an input. The **program complexity** is the complexity of computing the model when the fact-base is fixed but the rules-base is an input.

Claim 5: AIR-closure computation is PTime-complete in data-complexity and ExpTime-complete in program-complexity.

5 AIR and Logic Programming

Logic programming is a very popular declarative method of knowledge representation and programming. Like LPs, we define rules in AIR. However, AIR rules have different semantics from the LP rules, and they can be nested under one another. Different nesting of the same rules may yield (semantically) different AIR programs. In this section we investigate logic programming through AIR.

An LP rule is said to be **safe** if every variable occurring in the head of the rule or in a negative literal in the body also occurs in a positive literal in the body. We

```
@forAll :X, :Y .
:r a air:BeliefRule ;
    air:if { :X :conn :Y .} ;
    air:then [ air:rule :b1 ], [air:rule :b2 ] ;
    air:then [ air:rule :rAux ] .

:rAux a air:BeliefRule ;
    air:if { :X :aux :Y } ;
    air:else [ air:assert { :X :good-conn :Y . } ] .

:b1 a air:BeliefRule ;
    air:if { :X a :Infected . } ;
    air:then [ air:assert { :X :aux :Y } ] .

:b2 a air:BeliefRule ;
    air:if { :Y a :Infected . } ;
    air:then [ air:assert { :X :aux :Y } ] .
```

Fig. 7. Rewriting an example LP rule, in AIR : Connection between X and Y is good if both X and Y are not infected

can encode any safe LP rule $A \leftarrow A_1, \ldots, A_n, \mathit{not}(B_1), \ldots, \mathit{not}(B_m)$ in AIR by rewriting it as:

$$
\begin{aligned}
&A \leftarrow A_1, \ldots, A_n, \mathit{not}(\texttt{Aux}) \ .\\
&\mathit{Aux} \leftarrow B_1 \ .\\
&\ldots\\
&\mathit{Aux} \leftarrow B_m \ .
\end{aligned}
$$

Aux is a new predicate and its arguments are all the variable terms that appear in the literals of the rule's body. For example following rule : `good-conn(?X, ?Y)` $\leftarrow$ `conn(?X, ?Y)`, *not*`(infected(?X))`, *not*`(infected(?Y)).`, can be expressed as AIR rule as shown in Figure 7. Note that the predicate symbol `S` is translated to `:S`, and variable symbol `?S` is translated to `:S`, as quantified by an `@forAll` directive. We have translated binary predicates to property assertions, and the unary predicate to type assertion. Higher arity N-ary predicate $p(t_1, \ldots, t_n)$ can be encoded in N3 as {`(` $t_1 \ldots t_n$`) p true`}.

The nesting of rules impacts the order in which the rules are fired, and they may be nested properly to get the desired semantics. Since for Positive LPs (PLPs) the order in which the rules fire does not matter, the rules in PLP can be translated into AIR and included as top rules in an AIR program to get a semantically equivalent AIR program. When a logic language (e.g., LP) can be encoded in AIR with unchanged semantics, we will say that it can be **losslessly rewritten** into AIR.

We have seen that PLPs can be losslessly rewritten into AIR. OWL 2 RL[10] inference rules are all positive, and therefore they can be encoded in AIR and used concurrently with other AIR rules. Next we show that PSNHLPs can be losslessly rewritten into AIR.

Claim 6: PSNHLPs can be losslessly rewritten into AIR.
Proof. Let $P = P_1^+ \cup P_1^- \cup \ldots \cup P_n^+ \cup P_n^-$ be the PSNH-stratification of the PSNHLP P, and each P_i^- has n_i^- rules and P_i^+ has n_i^+ rules. Let the k^{th} rule in stratums P_i^+ and P_i^- be encoded in AIR with IDs $\texttt{:}r_{i_k}^+$ and $\texttt{:}r_{i_k}^-$ respectively. Then the nesting of rules shown in Figure 8 yields a lossless translation of P. This nesting can be generated programmatically for any PSNHLP P. `owl:Thing rdfs:subClassOf owl:Nothing` is a tautologically false statement.

[10] `http://www.w3.org/TR/owl2-profiles/#OWL_2_RL`

```
:rs a air:RuleSet ;
      air:rule : $r^+_{1_1}$, . . . , : $r^+_{1_{n_1}}$ ;
      air:rule : $r^-_{2_1}$, . . . , : $r^-_{2_{n^-_2}}$ ;
      GET_RULE_PROPERTY(2) .,
  where GET_RULE_PROPERTY(j) function is defined recursively as:
when $j < n$:
      air:rule [
            air:if { owl:Thing rdfs:subClassOf owl:Nothing . } ;
            air:else [ air:rule : $r^+_{j_1}$ ], . . ., [ air:rule : $r^+_{j_{n^+_j}}$ ] ;
            air:else [ air:rule : $r^-_{j+1_1}$ ], . . ., [ air:rule : $r^-_{j+1_{n^-_{j+1}}}$ ] ;
            air:else [ GET_RULE_PROPERTY($j + 1$) ] ]
when $j = n$ :
      air:rule [
            air:if { owl:Thing rdfs:subClassOf owl:Nothing . } ;
            air:else [ air:rule : $r^+_{j_1}$ ], . . ., [ air:rule : $r^+_{j_{n^+_j}}$ ] ]
```

Fig. 8. Rewriting PSNHLP in AIR

The rules in P^-_{i+1} and P^+_i are activated after $stage_{i-1}$. Since all the ground facts for predicates of level less than $\overline{i+1}$ in $PM(P)$ are inferred before $stage^-_i$, rules in P^-_{i+1} do not fire incorrectly in $stage^-_i$. □

Since non-recursive datalogs are PSNHLPs, they can be losslessly rewritten into AIR. The same can be said about SPARQL queries.

Claim 7: SPARQL SELECT *and* CONSTRUCT queries, without the query modifiers like ORDER BY and LIMIT, can be losslessly rewritten into AIR, and executed by the AIR reasoner.

Proof sketch. SPARQL SELECT *and* CONSTRUCT queries, without the query modifiers, can be translated to a non-recursive Datalog, Π_Q, with NAF under the answer set semantics [16]. Using claim 6 we can say that Π_Q can be losslessly rewritten into AIR. □

We have seen that PSNHLPs can be losslessly rewritten into AIR. However, more general stratified LPs cannot be translated into AIR. For example, the following rules cannot be losslessly rewritten into AIR for arbitrary facts of predicates P, Q, and S:

$$\mathrm{P(x, z)} \leftarrow \mathrm{P(x, y), P(y, z)}, \mathit{not}(\mathrm{Q(x, z)}) .$$
$$\mathrm{R(x, y)} \leftarrow \mathrm{S(x, y), not(P(x, y))} .$$

However, that can be resolved through a simple extension of AIR, whereby **RuleSet**s can be nested – `:rs`$_1$ `air:hasHigherPriority :rs`$_2$ – with the following meaning: the rules nested under **RuleSet** `:rs`$_2$ are activated only after the fixpoint is reached for the rules under `:rs`$_1$. Note that the rules under `:rs`$_1$

remain active. The corresponding change in reasoning algorithm is straightforward. This extension has some similarity with salience in Jess.

Claim 8: Stratified LPs can be losslessly rewritten into AIR extended with the nesting of **RuleSet**s.

6 Conclusions and Future Work

In this paper we have analyzed AIR, a rule language for the Semantic Web. AIR supports non-monotonic negation, and we have provided a declarative semantics that supports this negation by translating AIR programs to a specialized class of (locally) stratified LPs – PSNHLPs. While AIR does not support well-founded negation and is less expressive than other rule systems, its ability to construct explanations, declaratively manipulate them, and its support for scoped contextualized reasoning (SCR) make it sufficiently unique and useful for Web reasoning. As AIR is a language for the Web, SCR is especially relevant. AIR also allows for the nesting of rules; this allows for the segmentation of the conditions of a rule so that only part of them are revealed in the justifications. We have shown that nesting can also be leveraged to order rules and therefore encode fairly expressive LPs such as PSNHLP. Stratified LPs can be encoded in AIR with an incremental modification to AIR. Finally, we have shown that SPARQL queries may be executed by the AIR reasoner.

We plan to investigate ways to increase the expressiveness of AIR in the future. This is a twofold task. We have shown that PSNHLP can be encoded in AIR, but do not think that this is the most expressive LP that can be expressed in AIR. It may also be possible to investigate an alternative AIR reasoning algorithm in which actions that fire under the assumption of a failed condition are retracted when the rule's conditions match later on. Furthermore, with a well-defined LP translation of AIR program, it is possible to develop goal-based query answering mechanisms for these programs. We are also interested in investigating contextualization more closely.

References

1. Angele, J., Moench, E., Staab, S., Wenke, D.: Ontology-based query and answering in chemistry: Ontonova @ project halo. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 913–928. Springer, Heidelberg (2003)
2. Apt, K.R.: Logic programming. In: Handbook of Theoretical Computer Science, Formal Models and Sematics, vol. B (1990)
3. Bassiliades, N., Antoniou, G., Governatori, G.: Proof explanation in the dr-device system. In: Marchiori, M., Pan, J.Z., Marie, C.d.S. (eds.) RR 2007. LNCS, vol. 4524, pp. 249–258. Springer, Heidelberg (2007)
4. Berners-lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3logic: A logical framework for the world wide web. Theory Pract. Log. Program. 8(3) (2008)
5. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. In: IEEE Conference on Computational Complexity (1997)

6. Forgy, C.: RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1) (September 1982)
7. Forgy, C.L.: OPS5 users manual. In: Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University (1981)
8. Grosof, B.N.: SILK: Higher level rules with defaults and semantic scalability. In: Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems, RR 2009 (2009)
9. Grosof, B.N., Gandhe, M.D., Finin, T.W.: Sweetjess: Inferencing in situated courteous ruleml via translation to and from jess rules. In: Proceedings of the ISWC '02 International Workshop on Rule Markup Languages for Business Rules on the Semantic Web (2003)
10. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and ruleml. Technical report, W3C (2004)
11. Kagal, L., Hanson, C., Weitzner, D.: Using dependency tracking to provide explanations for policy management. In: IEEE International Workshop on Policies for Distributed Systems and Networks (2008)
12. Kagal, L., Jacobi, I., Khandelwal, A.: Gasping for AIR - why we need linked rules and justications on the semantic web. In: Under review at the International Semantic Web Conference, ISWC 2010 (2010)
13. Khandelwal, A., Bao, J., Kagal, L., Jacobi, I., Ding, L., Hendler, J.: Analyzing the AIR language: A semantic web rule language. Technical Report, Department of Computer Science, Rensselaer Polytechnic Institute (2010)
14. Kifer, M.: Rule interchange format: The framework. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 1–11. Springer, Heidelberg (2008)
15. Liang, S., Fodor, P., Wan, H., Kifer, M.: Openrulebench: an analysis of the performance of rule engines. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009 (2009)
16. Polleres, A.: From SPARQL to rules (and back). In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007 (2007)
17. Polleres, A., Feier, C., Harth, A.: Rules with contextually scoped negation. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 332–347. Springer, Heidelberg (2006)
18. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. Technical report, W3C (2006)
19. Przymusinski, T.C.: On the declarative semantics of deductive databases and logic programs. In: Foundations of Deductive Databases and Logic Programming (1988)
20. Schenk, S., Staab, S.: Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In: Proceeding of the 17th International Conference on World Wide Web, WWW 2008 (2008)
21. Schor, M.I., Daly, T., Lee, H.S., Tibbitts, B.: Advances in RETE pattern matching. In: AAAI (1986)

Query-Based Access Control for Ontologies

Martin Knechtel[1,⋆] and Heiner Stuckenschmidt[2]

[1] SAP Research Center Dresden, Germany
martin.knechtel@sap.com
[2] Computer Science Institute, University of Mannheim, Germany
heiner@informatik.uni-mannheim.de

Abstract. Role-based access control is a standard mechanism in information systems. Based on the role a user has, certain information is kept from the user even if requested. For ontologies representing knowledge, deciding what can be told to a user without revealing secrets is more difficult as the user might be able to infer secret knowledge using logical reasoning. In this paper, we present two approaches to solving this problem: query rewriting vs. axiom filtering, and show that while both approaches prevent the unveiling of secret knowledge, axiom filtering is more complete in the sense that it does not suppress knowledge the user is allowed to see while this happens frequently in query rewriting. Axiom filtering requires that each axiom carries a label representing its access level. We present methods to find an optimal axiom labeling to enforce query-based access restrictions and report experiments on real world data showing that a significant number of results are retained using the axiom filtering method.

1 Motivation

Access control is an essential operation in standard information systems to prevent unauthorized access and use of information from the system. In a traditional information system, where all the available information is stored explicitly, it is possible to simply label information items with the roles, a user must have, to be allowed to receive this particular information. With knowledge represented in ontologies, this approach does not work anymore, because new knowledge can be derived, leading to the *'inference problem'* [5]: avoiding a situation where a user can infer knowledge he should not have access to using knowledge he is allowed to access. To make the problem well defined, we assume that the user has the same ability to derive knowledge as the system.

In this paper, we compare two existing proposals for solving the inference problem: query rewriting vs. axiom filtering. For both, we start from an access restriction given in the form of a query, whose result is a set of axioms that shall be protected. Such a query could, for example, address knowledge about a concept and all subconcepts in order to restrict knowledge along the subsumption hierarchy comparable to information systems restricting access to files in a directory and all subdirectories. Conflict resolution mechanism might be necessary then since a concept might have multiple superconcepts. The

⋆ This research was partly funded by the German Federal Ministry of Economics and Technology under the promotional reference 01MQ07012.

query rewriting approach proposed in [4] is based on the idea of rewriting user queries based on the role a user has in such a way that the result to the rewritten query only returns knowledge the user is allowed to see. The axiom filtering approach proposed in [1] assumes an a priori labeling of axioms in the ontology to consistently derive labels for implicit consequences. Axioms and consequences are delivered based on a comparison of user label and axiom label. Our assessment of the two approaches concludes that axiom filtering is independent of the ontology language and more complete in the sense that it does not suppress knowledge the user is allowed to see.

However axiom filtering requires an a priori labeling of axioms and it is not clear from previous work how to create an access labeling from query-based access restrictions. Our main contributions are (1) algorithms to repair a given axiom labeling in an optimal way so that a query-based access restriction is enforced to explicit and implicit knowledge, (2) conflict resolution strategies for cases where query-based access restrictions contain conflicts, (3) empirical results for our algorithms with practical ontologies. Our main result is that axiom filtering provides higher availability of knowledge compared to query rewriting.

2 Preliminaries

2.1 Ontologies

Ontologies are formal descriptions of the terminology used in an application domain. A number of logical languages have been proposed for representing ontologies. In this paper, we only consider sublanguages of the Web Ontology Language (OWL) that can be translated to Description Logics (DL).

Formally, an *ontology* O is a finite set, whose elements are called *axioms*, such that every subset of an ontology is itself an ontology. If $O' \subseteq O$ and O is an ontology, then O' is called a *sub-ontology* of O. One can distinguish ABox axioms A and TBox axioms T and let $O = T \cup A$. An ontology language specifies which sets of axioms are admitted as ontologies. For instance, given a Description Logic $\mathcal{L}$ (e.g., the DL $\mathcal{SHOIN}(\mathbf{D})$ underlying OWL DL), an ontology is a finite set of general concept inclusion axioms (GCIs) of the form $C \sqsubseteq D$, concept assertion axioms of the form $C(a)$ and role assertion axioms of the form $R(a, b)$ for $\mathcal{L}$-concept descriptions C, D, role R and individuals a, b. In order not to mix user roles and DL roles, we stick to the OWL lingo and call DL roles from now on *properties*. The signature $sig(O)$ of an ontology is the set of all concept and role names occurring in its axioms. Given an ontology language, a *monotone consequence relation* $\models$ is a binary relation between ontologies O and *consequences* c such that if $O \models c$, then for every ontology $O' \supseteq O$ it holds that $O' \models c$. If $O \models c$, we say that *c follows from* O or that O *entails* c. Often, a consequence c already follows from a subset $S \subseteq O$ of the axioms in the ontology. We call such a subset an explanation for $O \models c$ if there is no subset $S' \subset S$ such that $S' \models c$. Note that for one consequence there might be multiple explanations.

A query to an ontology is a conjunction $Q = A_1, \cdots, A_n$ of OWL axioms over sig(O), but not necessarily from O, containing variables. For a concrete definition of the form of axioms see [12]. The set of variables occurring in Q is denoted as $var(Q)$. Let $ind(O)$ be the set of individuals in O, then the result of a query is the set of all mappings $\mu : var(Q) \rightarrow ind(O)$ assigning individuals from O to variables in Q. An answer $\mu(Q)$ to a query Q is an instantiation of all variables in the query, so that $O \models \mu(Q)$ [12]. Note that there might be several possible μ for one query.

2.2 Access Control

Access control systems enable the regulation of access to protected resources (i.e. objects) in distributed systems by subjects such as users or system processes. They can be categorized in discretionary access control (DAC), mandatory access control (MAC), and role-based access control (RBAC) models. In DAC-based systems, the permissions to access an object are defined by its owner. In MAC models, the system determines the access to objects either by utilizing access rules or lattices for assigning permissions to subjects. It thus removes the ability of the users to control access to their resources. RBAC systems finally remove the explicit use of subjects within access rules or lattices and replace them with roles, which form a logical group of a number of subjects. In fact, permissions are assigned to roles and the subjects are assigned members of a number of roles. Thus changes of single subjects do not necessarily have consequences in the actual access control policies. On the most fine-grained level, permissions can be defined on the level of axioms, or on the level of query responses.

2.3 Access Restrictions as Queries

Assume we want customers and employees to query knowledge from a product ontology. From Example 1, employees have full access and we do not want customers to see if any product gets an increased price soon. This restriction could be defined by enumerating all query responses except the price increase as permissions and assigning them to the respective user role. There are two problems with this approach. First of all, the price increase can still be inferred if the axioms of O can be queried. Further, enumerating all query responses, however, is not feasible in practice and asks for more efficient ways of specifying these restrictions, e.g. by means of a query.

Example 1. Let O be an ontology from a marketplace in the Semantic Web with the following axioms

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

The consequence $c_1 : \mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$ follows from each of the explanations $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}, \{a_1, a_3, a_5\}$. The consequence $c_2 : \mathit{LowProfitService}(\mathit{ecoCalculatorV1})$ follows from each of the explanations $\{a_1, a_2\}, \{a_1, a_3\}$. Three more instance assertions of individual $\mathit{ecoCalculatorV1}$ to the concept names $\mathit{EUecoService}, \mathit{HighperformanceService}, \mathit{ServiceWithLowCustomerNr}$ are consequences of O.

A way is to define permissions intentionally in terms of queries over the signature of the ontology. More specifically, we can describe facts that should not be accessible by a certain role in terms of a set of axioms - the same kinds of axioms used in queries - whose instantiations should not be derivable from query results. In the case of the example above, we could formulate the following access restriction for customers:

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

stating that for no instantiation of the variable x it should be possible to infer that it is an instance of $\mathit{ServiceWithComingPriceIncrease}$.

3 Enforcing Access Restrictions

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

3.1 Access Control by Query Rewriting

One option for enforcing access restrictions is by means of query rewriting. This approach has been proposed in [4] as a suitable way for enforcing access restrictions in the context of SPARQL queries, while the TBox is assumed to be completely public. Similar approaches are also allowing to hide TBox parts [7], or to define not the restrictions but the permissions by a query [3]. The idea in [4] is to automatically add filter conditions to the query that suppress such answers the user is not supposed to see. Given a Query Q and a set of access restrictions $\{AR_1, \cdots, AR_n\}$ that apply to the current user, the query can be rewritten to a new query that is defined as:

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

Where the junction of two queries $Q_1 \wedge Q_2$ is the junction of all contained query axioms $\bigwedge_{q \in Q_1} q \wedge \bigwedge_{q \in Q_2} q$ [12]. This way of rewriting the query based on the access restrictions of the individual users effectively prevents the system from giving away restricted knowledge. In particular, using query rewriting, the effective answer to a query is

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

It however comes with a problem: it hides more knowledge than necessary. In particular, in the example above where we want to hide from customers that some product is increased in price, the query rewriting approach hides too much knowledge. If a customer for instance asks the system for all high performance services, thus $Q = \mathit{HighperformanceService}(x)$, this query will be rewritten to $\mathit{HighperformanceService}(x) \wedge \neg \mathit{ServiceWithComingPriceIncrease}(x)$. This query will only return high performance services which will not be increased in price. This is unfortunate, because the knowledge that $\mathit{ecoCalculatorV1}$ is a high performance service was not supposed to be hidden. Similarly querying for instances of the remaining four concept names in $sig(O)$ are filtered, resulting in five queries without an answer.

3.2 Access Control by Axiom Filtering

A framework to control access to an ontology's axioms is introduced in [1]. In contrast to the query rewriting approach above, the TBox is not assumed to be completely public. The idea is to label each axiom with a certain access restriction. Users are labeled with the restrictions they are allowed to see. The approach is to use a labeling lattice $(L, \leq)$; i. e. a set L of labels together with a partial order $\leq$ such that every finite set of labels has a join ($\oplus$, supremum, least upper bound) and a meet ($\otimes$, infimum, greatest lower bound) w.r.t. $\leq$. Every axiom a in the ontology O is assumed to have a label $\mathsf{lab}(a) \in L$, and each user receives also a label $\ell \in L$. The sub-ontology to which a user with label ℓ has access is defined as

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

The sub-ontologies $O_{\ngeq \ell}, O_{\nleq \ell}$ etc. can be defined analogously. Applied to our scenario with the user roles customer (ℓ_C) and employee (ℓ_E), let the labeling lattice be $(L, \leq)$ with $L = \{\ell_C, \ell_E\}$ and $\leq = \{(\ell_E, \ell_C)\}$. Let the labeling function lab assign label ℓ_C to axioms a_1, a_2, a_3 and ℓ_E to axioms a_4, a_5. Employees can see $O_{\geq \ell_E} = \{a_1, a_2, a_3, a_4, a_5\}$, i.e. the complete ontology. Customers can see $O_{\geq \ell_C} = \{a_1, a_2, a_3\}$. Intuitively, the access restriction to a consequence, called boundary, should be based on the access restriction of its implying axioms. The access restriction for a consequence with multiple explanations should be the least restrictive of all explanations and within one explanation the most restrictive of all axioms. Formally, a consequence c with n explanations $S_1, \ldots, S_n$ has boundary $\bigoplus_{i=1}^{n} \bigotimes_{a \in S_i} \mathsf{lab}(a)$. In our example, each of the four explanations for c_1 has label $(\ell_C \otimes \ell_C \otimes \ell_E) = \ell_E$, thus the boundary is ℓ_E, i.e. employees can see it but customers not. Consequence c_2 has boundary ℓ_C, i.e. employees and customers can see it. Apart from c_1, c_2, instance relationships to the three remaining concepts in $sig(O)$ have boundary ℓ_C as can be verified easily. A customer querying for instances of the five concept names in the ontology will get no answer for $Q = \mathit{ServiceWithComingPriceIncrease}(x)$ but will get an answer for the four remaining queries. So axiom filtering provides $4/5$ answers, while query rewriting provides $0/5$ answers.

3.3 Discussion

As we have seen, query rewriting and axiom filtering are approaches of ensuring that no classified knowledge is given to users that do not have the permission to see it. Both approaches do neither require to track the history of queries nor disallow query askers of the same user role to share any knowledge. We have seen that query rewriting is suboptimal with respect to availability in the sense of preserving maximal access to non-restricted knowledge. Axiom filtering provides a higher availability and is more general since it is independent of the concrete ontology language which makes the approach preferable in many situations. However it requires an a priori axiom labeling, and it is not clear how to enforce query-based access restrictions. Previous work on labeled ontologies focused on computing a consequence's label based on axiom labels [1] and on repairing the axiom labeling in order to determine one consequence's label [9,10]. However, access restrictions in the form of queries might require changing

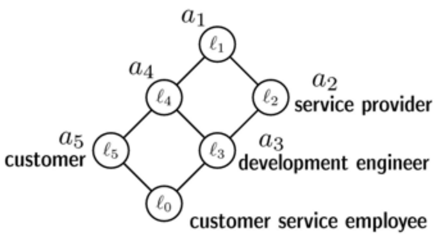


Fig. 1. Lattice $(L, \leq)$ with 4 user labels and an assignment of 5 axioms to labels

labels of multiple consequences simultaneously. Such a mechanism will be presented in the next section. Our main quality criterion for the algorithms is availability. In the empirical evaluation we measure how many knowledge is additionally accessible with axiom filtering compared to query rewriting.

4 Optimal Axiom Labeling for Implementing Access Control

In the last section we have only shown that there is an axiom labeling to enforce access restrictions for a selected example. Now we will elaborate how to compute it in general. We are starting from an arbitrary label assignment, and change it in a minimal way so that a given access restriction is enforced.

Example 2. We continue Example 1. Let $(L, \leq)$ be the lattice shown in Figure 1, where valid user labels are $\ell_0, \ell_2, \ell_3, \ell_5$ which represent user roles as illustrated. The condition for a valid user label is the join prime property discussed in [1]. Let O of Example 1 be a labeled ontology where the function lab assigns to each axiom a_i the label ℓ_i as shown in Figure 1. The computed boundary is ℓ_3 for c_1, since $= (\ell_1 \otimes \ell_2 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_2 \otimes \ell_5) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_5)$. It is ℓ_2 for c_2, since $= (\ell_1 \otimes \ell_2) \oplus (\ell_1 \oplus \ell_3)$. For users ℓ_0 and ℓ_3, consequences c_1 and c_2 are visible. For user ℓ_2, only c_2 is visible.

We now define a notion for changing an axiom label assignment. Beforehand, we define the function lbl in order to address computed boundaries of consequences in a convenient way.

Definition 1 (Consequence Labeling Function). *Let O be a labeled ontology, $(L, \leq)$ a labeling lattice, $\mathsf{lab} : O \to L$ a labeling function. The* consequence labeling function *$\mathsf{lbl} : \{c \mid O \models c\} \to L$ assigns labels to consequences and is defined as $\mathsf{lbl}(c) =$ computed boundary of c.*

Definition 2 (MCS). *Let O be an ontology, c any consequence of O, $(L, \leq)$ a lattice, lab a labeling function, G a set of goals of the form (c, ℓ_g) with goal label ℓ_g for consequence c, M a set of assignments (a, ℓ) of label ℓ to axiom a. The* modified assignment *lab_M is defined to be*

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

The respective consequence labeling function lbl_M *is given by Definition* [1]*. The set* M *is called* multiple change set *(MCS) iff for any* $c, (c, \ell_g) \in G : \mathsf{lbl}_M(c) = \ell_g$ *and there is no* $M' \subset M$ *with* $\mathsf{lbl}_{M'}(c) = \ell_g$*.*

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

4.1 Computing a Change Set for One Goal Label

If G is the singleton set of only one tuple (c, ℓ), computing a multiple change set boils down to computing a change set (CS) which has been introduced in our prior work in [10,9]. For every CS $S \subseteq O$ there is a MCS $M := \{(a, \ell_g) \mid a \in S\}$ and $\mathsf{lbl}_M(c) = \ell_g$ holds. The computation of a CS exploited main ideas from axiom-pinpointing [8,2] and we presented a black-box approach that yields the desired set. Intuitively, a consequence c needs to be made more public if $\ell_g > \mathsf{lbl}(c)$ or less public if $\ell_g < \mathsf{lbl}(c)$. From the perspective of the target users who see $O_{\geq \ell_g}$, the former is achieved by including an axiom set IAS to their ontology and the latter by removing an axiom set RAS from other user's ontologies. The definition of an IAS (RAS) is a generalization of the definition of a MinA (diagnosis) [10].

Definition 3 (IAS,RAS). *A* minimal inserted axiom set *(IAS) for* ℓ_g *is a subset* $I \subseteq O_{\not\geq \ell_g}$ *such that* $O_{\geq \ell_g} \cup I \models c$ *and for every* $I' \subset I : O_{\geq \ell_g} \cup I' \not\models c$*. A* minimal removed axiom set *(RAS) for* ℓ_g *is a subset* $R \subseteq O_{\not\leq \ell_g}$ *such that* $O_{\not\leq \ell_g} \setminus R \not\models c$ *and for every* $R' \subset R : O_{\not\leq \ell_g} \setminus R' \models c$*.*

A CS is either an IAS, a RAS, or union of both. As elaborated in [10], computing IAS and RAS is tightly related to computing explanations (also called MinA) and diagnoses. The computation by a Hitting Set Tree (HST) algorithm [11] is repeated here only briefly. The HST algorithm makes repeated calls to an auxiliary procedure that computes one CS. A tree is built, where each node is labeled with a CS and each edge with an axiom. If the CS labeling a node has n axioms ($S := \{a_1, \ldots, a_n\}$), then this node is expanded with n children: the edge to the i-th child labeled with a_i, the child labeled with a CS that is not allowed to contain neither a_i nor any ancestor's edge label. This ensures that each node is labeled with a CS distinct from those of its predecessors.

HST optimizations such as *early termination* and *node reuse* avoid redundant computations and are included in current implementations. Another optimization is putting a *cardinality limit*, applicable when not all, but only the CS of minimal cardinality $|S|$ is of interest. Then nodes might contain partial solutions, called *partial CS*, in the sense that some axioms are missing, but still the smallest CS is proven to be found [10,9].

Example 3. We continue Example [2]. Assume we want to make c_1 as private as possible, i.e. $G = \{(c_1, \ell_0)\}$. All RAS are $\{a_1\}, \{a_2, a_3\}, \{a_4, a_5\}$, so the smallest MCS is $M_1 = \{(a_1, \ell_0)\}$ and we get $\mathsf{lbl}_{M_1}(c_1) = \ell_0$. As second example assume we want to make c_2 as public as possible, i.e. $G = \{(c_2, \ell_1)\}$. All IAS are $\{a_2\}, \{a_3\}$, so one of the smallest MCS is $M_2 = \{(a_3, \ell_1)\}$ and we get $\mathsf{lbl}_{M_2}(c_2) = \ell_1$.

Algorithm 1. Extract cMCS with optimizations CS reuse (switch off: remove Line 11) and cardinality limit (switch off: in Line 7 replace "$n - |M|$" by "∞")

```
Procedure init-cMCS-extraction($O$, lab, $(L, \leq)$, $G$)
Input: $O$, lab: labeled ontology; $(L, \leq)$: lattice; $G$: goal set
 1: Global: $O, \mathsf{lab}, G' := \{(c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, C_S) \mid (c, \ell_g) \in G,$
      $\mathrm{is}_I := \ell_g \not< \mathsf{lbl}(c) \wedge O_{\geq \ell_g} \not\models c,$          (decision to compute IAS)
      $\mathrm{is}_R := \ell_g \not> \mathsf{lbl}(c) \wedge O_{\not\leq \ell_g} \models c,$          (decision to compute RAS)
      $C_S := \emptyset\}$                                              (reuse set for CS)
Procedure extract-partial-cMCS($K$, $n$)
Input: $K$: prohibited label changes; $n$: cardinality limit
Output: first $n$ elements of a cMCS
 1: $M := \emptyset$
 2: for each goal $(c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, C_S) \in G'$ do
 3:     $H := \{a \mid (a, \ell_g) \in K\}$           (set of axioms not allowed to be labelled with $\ell_g$)
 4:     if $\exists S' \in C_S : \emptyset = S' \cap H$ then
 5:         $S := S'$                                               (CS reuse)
 6:     else
 7:         $S :=$ extract-partial-CS($O, \mathsf{lab}, c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, H, n - |M|$)      (defined by [9])
 8:         if $\emptyset = S$ then
 9:             return $\emptyset$      (HST normal termination for one goal fires for complete goal set)
10:         if $|S| \neq n - |M|$ then
11:             $C_S := C_S \cup \{S\}$                       (remember only non-partial CS)
12:     $M := M \cup \{(a, \ell_g) \mid a \in S\}$
13: return $M$
```

4.2 Computing a Multiple Change Set for Multiple Goal Labels

An MCS for several goals consists of CS for each of the individual goals. However, it is no solution to compute single CS and combine them since this might not yield the smallest MCS or they might even conflict.

Example 4. We combine both goals of Example 3 simultaneously, i.e. we want to make c_1 as private as possible and c_2 as public as possible, $G = \{(c_1, \ell_0), (c_2, \ell_1)\}$. Just concatenating the above mentioned MCS to $M = M_1 \cup M_2 = \{(a_1, \ell_0), (a_3, \ell_1)\}$ is no MCS since $\mathsf{lbl}_M(c_2) = \ell_0 \neq \ell_1$. However, $M = \{(a_4, \ell_0), (a_5, \ell_0), (a_2, \ell_1)\}$ is an MCS.

For this reason we call any combination of CS a *candidate MCS* (cMCS). To compute the shortest MCS, we introduce Algorithm 2 which is similar to the HST algorithm for computing the shortest CS in [9]. The only difference is that each call to the auxiliary procedure computes a (partial) cMCS instead of a (partial) CS which is assigned to a node in the search tree, and edges are not labeled with an axiom but with a tuple (a, ℓ) which is not allowed in the child node's (partial) cMCS.

A (partial) cMCS is computed by a call extract-partial-cMCS(K, n) to the auxiliary procedure in Algorithm 1, where K is the set of prohibited label changes, i.e. all tuples at edges to ancestors in the HST, and n is the size of the currently known shortest MCS. The procedure comes with 2 optimizations: *CS reuse* and *cardinality limit*. As any

Algorithm 2. HST algorithm to find smallest MCS for G

```
Procedure hst-extract-smallest-MCS($(O, \mathsf{lab}, (L, \leq), G, K)$)
Input: $O$, lab: labeled ontology; $(L, \leq)$: lattice; $G$: goal set; $K$: prohibited label changes
Output: MCS of minimum cardinality
 1: Global $M_{\text{min}} := \emptyset, n := \infty, G$
 2: init-cMCS-extraction($(O, \mathsf{lab}, (L, \leq), G)$)
 3: expand-hst-MCS($(K)$)
 4: return $M_{\text{min}}$
Procedure expand-hst-MCS($(K)$)
Input: $K$: prohibited label changes
Side effects: modifications to $M_{\text{min}}$ and $n$
 1: $M :=$ extract-partial-cMCS($(K, n)$)
 2: if $M = \emptyset$ then
 3:     return                                          (HST normal termination)
 4: if $|M| < n$ then
 5:     if $(a, \ell_1), (a, \ell_2) \in M \implies \ell_1 = \ell_2$ then
 6:         if $\forall (c, \ell_g) \in G : \mathsf{lbl}_M(c) = \ell_g$ then
 7:             $M_{\text{min}} := M$
 8:             $n := |M_{\text{min}}|$
 9:         else
10:             ...                                    (semantic conflict resolution)
11:     else
12:         ...                                        (syntactic conflict resolution)
13: for the first $(n-1)$ label changes $(a, \ell) \in M$ do
14:     expand-hst-MCS($(K \cup \{(a, \ell)\})$)
```

cMCS is a combination of CS, one CS might be contained in several cMCS. Instead of computing it anew for every cMCS, the first optimization reuses it. Putting a cardinality limit is a second optimization which computes a cMCS or stops once this has reached a size n and returns a potentially partial cMCS. Computing partial CS for one goal turned out to reduce execution time [9]. In a partial cMCS, the last contained CS is partial. Partial CS are not reused.

Turning to Algorithm 2, whenever a cMCS M is found with $|M| < n$, it is shorter than our currently known shortest MCS and we can be sure that it is not partial. The question remains if it is a MCS or only a cMCS, which is checked in Line 6: neither is an axiom allowed to have multiple labels assigned (*syntactic conflict*) nor might a change set for one goal influence any other goal which is the case if any computed boundary does not equal the goal label (*semantic conflict*). Only after passing both checks, we update our globally known shortest known MCS M_{min} in Line 7. Loosening the constraints of a goal set, the semantic conflicts can be resolved in Line 10 or syntactic conflicts can be resolved in Line 12 which is explained in the next section.

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

Theorem 1 (Cardinality Limit Optimization). *Let O,* lab *be a labeled ontology and G a goal set. If m is the minimum cardinality of all MCS for G, the HST Algorithm 2 outputs a MCS M such that $|M| = m$.*

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

4.3 Conflict Resolution

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

Example 5. We continue Example 2. Assume $G = \{(c_1, \ell_4), (c_2, \ell_3)\}$. For the goal (c_1, ℓ_4) all IAS are $\{a_2\}, \{a_3\}$. For the goal (c_2, ℓ_3) all RAS are $\{a_1\}, \{a_2\}$. The cMCS $M_1 = \{(a_2, \ell_4), (a_2, \ell_3)\}$ is obviously no MCS due to a syntactic conflict. But also the remaining cMCS $M_2 = \{(a_2, \ell_4), (a_1, \ell_3)\}, M_3 = \{(a_3, \ell_4), (a_1, \ell_3)\}, M_4 = \{(a_3, \ell_4), (a_2, \ell_3)\}$ are no MCS due to semantic conflicts, since $\mathsf{lbl}_{M_2}(c_1) = \mathsf{lbl}_{M_3}(c_1) = \ell_3 \neq \ell_4$ and $\mathsf{lbl}_{M_4}(c_2) = \ell_4 \neq \ell_3$.

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

1. Overrestrictive: accept lower labels for a minimal number of consequences than specified by the goal label. Formally, $\forall (c, \ell_g) \in G : \mathsf{lbl}_M(c) \neq \ell_g \implies \mathsf{lbl}_M(c) < \ell_g$ and cardinality $|\{(c, \ell_g) \in G \mid \mathsf{lbl}_M(c) \neq \ell_g\}|$ is minimal. Applied to the above example, $\{(a_2, \ell_3)\}$ is a RMCS.
2. Overpermissive: accept higher labels for a minimal number of consequences than specified by the goal label. Formally, $\forall (c, \ell_g) \in G : \mathsf{lbl}_M(c) \neq \ell_g \implies \mathsf{lbl}_M(c) > \ell_g$ and cardinality $|\{(c, \ell_g) \in G \mid \mathsf{lbl}_M(c) \neq \ell_g\}|$ is minimal. Applied to the above example, $\{(a_2, \ell_4)\}$ is a RMCS.

Algorithm 3. Computing a RMCS, overpermissive strategy (for overrestrictive strategy: replace "$\oplus$" with "$\otimes$" in Line 3, "$\geq$" with "$\leq$" in Line 4, "$>$" with "$<$" in Line 5)

Basis is the Algorithm 2. In Procedure hst-extract-smallest-MCS, add global variables $N := \emptyset, r := \infty$, and add before Line 4:

```
1: if $\emptyset = M_{\text{min}}$ then
2:     return $N$
```

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

```
1: $N' := M$
2: for each $a : (a, \ell_1), (a, \ell_2) \in N' \wedge \ell_1 \neq \ell_2$ do
3:     $N' := N' \setminus \{(a, \ell_1), (a, \ell_2)\} \cup \{(a, \ell_1 \oplus \ell_2)\}$
4: if $\forall (c, \ell_g) \in G : \mathsf{lbl}_{N'}(c) \geq \ell_g$ then          (fulfills overpermissive strategy)
5:     $r' := |\{(c, \ell_g) \in G \mid \mathsf{lbl}_{N'}(c) > \ell_g\}|$
6:     if $r' < r$ then
7:         $N := N'$
8:         $r := r'$
```

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

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

Algorithm 3 describes the resolution of syntactic conflicts. It is an adapted version of Algorithm 2, where additionally the global variable r stores the minimal number of overpermissive (overrestrictive) consequence labels and N stores the RMCS with minimal r. Again this Algorithm relies on the cMCS extraction Algorithm 1 and the optimization of reusing CS can be applied. The cardinality limit optimization is of no use here since if no MCS is found, then no cardinality limit is set and the HST is fully expanded.

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

Example 6. We continue Example 2. Assume $G = \{(c_1, \ell_2), (c_2, \ell_5)\}$. For the goal (c_1, ℓ_2) all IAS are $\{a_4\}, \{a_5\}$. For the goal (c_2, ℓ_5) all IAS are $\{a_2\}, \{a_3\}$, all RAS are $\{a_1\}, \{a_2, a_3\}$. Obviously no combination of CS for both goals yields a syntactic conflict. Nevertheless there is no MCS since every combination of CS has a semantic conflict. After conflict resolution, an overpermissive RMCS is

$N_{OP} = \{(a_4, \ell_2), (a_2, \ell_2 \oplus \ell_5 = \ell_1), (a_3, \ell_5\}$, yielding $\mathsf{lbl}_{N_{OP}}(c_1) = \ell_1, \mathsf{lbl}_{N_{OP}}(c_2) = \ell_1$. An overrestrictive RMCS is $N_{OR} = \{(a_4, \ell_2), (a_2, \ell_2 \otimes \ell_5 = \ell_0), (a_3, \ell_5)\}$, yielding $\mathsf{lbl}_{N_{OR}}(c_1) = \ell_5, \mathsf{lbl}_{N_{OR}}(c_2) = \ell_5$.

5 Experiments

We implemented and evaluated our algorithms empirically with large practical ontologies. The following sections describe our test setting and the results.

5.1 Test Procedure and Test Data

We test on a PC with 2GB RAM and Intel Core Duo CPU 3.16GHz. We implemented all approaches with Java 1.6, Pellet 2.0 and OWL API trunk revision 1150. As labeling lattice $(L, \leq)$ we use the one introduced in Figure 1. We use the top lattice element ℓ_1 for public knowledge, ℓ_2 for intermediate knowledge and ℓ_3 for top secret knowledge.

Our test ontologies O^{GEOM}[1], O^{MGED}[2], O^{PROCESS}[3] are selected ontologies from the TONES Ontology Repository[4] with a high number of individuals. At time of their download on March 25th 2010, they had the characteristics given in Table 1. The test ontology O^{FUNCT} is an OWL ontology for functional description of mechanical engineering solutions presented in [6].

In a first experimental setting we tested the availability of access control by query rewriting vs. access control by axiom filtering. Initially each ontology axiom is labeled ℓ_1 so that the complete ontology is public. This reflects default allow-all behavior of a security policy. Then for each concept C in the ontology, we apply access restriction $AR = C(x)$ by including each query result $c = \mu(AR)$ with goal label ℓ_3 in the goal set. The computed MCS is used to create a newly labeled ontology, on which we perform the following queries. We count for every C-instance the instance relationships to concepts other than C which are available for public users (ℓ_1). With query rewriting their count is 0. With axiom filtering their count is the availability gain of axiom filtering vs. query rewriting. For cMCS extraction defined by Algorithm 1, we tested both optimizations CS reuse and cardinality limit separately and their combination. In this setting every cMCS is automatically an MCS since there are no conflicting goals. Although not included in Algorithm 2 for transparency reasons, the mentioned usual HST optimizations *early termination* and *node reuse* are included in our implementation.

In a second experimental setting we tested conflict resolution strategies in cases where multiple goals conflict each other so that no MCS can be computed without relaxing one of the goals. We test the overrestrictive conflict resolution approach vs. the overpermissive conflict resolution approach of Algorithm 3 with the same ontologies. Only the CS reuse optimization of the auxiliary procedure in Algorithm 1 to extract cMCS is used, cardinality limit is not used for reasons explained in Section 4.3. First all axioms are labeled with intermediate security level, i.e. ℓ_2. A goal set is created for

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

Table 1. Test sets consisting of ontologies and goal sets

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

Table 2. Gained assertions compared to query rewriting, performance of optimizations

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

each concept C containing the same consequences described above, but now one half of this set has goal label ℓ_1 and the other half ℓ_3. Some of the resulting goal sets are contradictory. We test Algorithm 3 to compute a RMCS with overpermissive vs. overrestrictive conflict resolution strategy for the same goal set and we count the number of overpermissive/overrestrictive consequence labels.

For both experiments the test data characteristics are given in Table 1. The number of goal sets and of goals per goal set are the same for both experiments since they contain the assertions to each of the ontology's concepts, only with different goal labels. In order to limit runtime we compute in maximum 10 cMCS before the HST Algorithms 2 and 3 return, so there might be MCS or RMCS of lower cardinality.

5.2 Empirical Results

The experimental results for the first experiment are given in Table 2. It compares availability of access control by query rewriting vs. access control by axiom filtering and it compares performance of both optimizations cardinality limit vs. CS reuse. The given total number of CS includes reused CS. The number of cMCS is equal to the number

Table 3. Conflict resolution with overrestrictive (OR) strategy vs. overpermissive (OP) strategy

Test set	♯goal sets confl.	♯goals per confl. goal set	strat-egy	Results (averages per conflicting goal set)					
				♯cMCS	♯RMCS	|RMCS|	runtime (min-utes)	♯OR/OP cons. labs	% of enforced goals
O^{FUNCT}	19	50.3	OR	10.0	10.0	101.4	2.2	19.5	61%
			OP	10.0	10.0	110.0	2.0	20.3	60%
O^{GEOM}	39	150.7	OR	10.0	10.0	139.4	45.4	63.3	58%
			OP	10.0	10.0	140.4	37.0	52.1	65%
O^{PROCESS}	23	31.0	OR	10.0	10.0	32.3	0.9	12.7	59%
			OP	10.0	10.0	32.6	0.8	11.0	64%
O^{MGED}	16	165.8	OR	10.0	10.0	140.4	814.6	75.6	54%
			OP	10.0	10.0	141.6	780.8	51.9	69%

of MCS since the goals contain no conflicts with the first experiment. The number of gained assertions confirms that our ideas improve availability of knowledge when using axiom filtering instead of query rewriting. While the number of gained assertions is comparable between the optimizations applied, their runtime differs significantly. CS reuse alone, and also in combination with cardinality limit runs significantly faster compared to using cardinality limit optimization only. Testing O^{MGED} with cardinality limit optimization did not terminate after 4 days, so no results are provided.

The experimental results for the second experiment comparing conflict resolution with overrestrictive strategy vs. overpermissive strategy are given in Table 3. Only some of the goal sets constructed as described above are conflicting, and results are only given for those. Only the given percentage of the goals in one goal set are enforced, the remaining consequences have overpermissive/overrestrictive labels making them more public/private than intended by the goal set. The runtime limit of 10 cMCS was hit in every case, making the HST algorithm stop so there might be RMCS with less overpermissive/overrestrictive consequence labels when relaxing this runtime limit.

6 Conclusions

We considered scenarios where different parts of a given ontology should be visible for different users. We introduced access restrictions intentionally defined by means of a query. The answer to that query is the set of those axioms and consequences of the ontology, which have to be access restricted. We compared two basic approaches to enforce those access restrictions: query rewriting vs. axiom filtering. Compared to query rewriting, axiom filtering allows higher availability in the sense of more answers delivered to a user without unveiling any secret and is independent of any ontology language.

Axiom filtering relies on an axiom labeling. The problem solved by this paper is to find an optimal axiom labeling to enforce given access restrictions. Given a query-generated goal set containing consequences and intended labels, our algorithms compute a minimal change set defining a new axiom labeling. We show that a change set does not always exist since a goal set might contain conflicts, and we provide 2 conflict resolution strategies to relax the goal set so that a change set can be computed. Our experimental results show that our algorithms behave well in practical scenarios.

As future work we will look at other criteria for the minimality of change sets for example not counting the amount of changed axiom labels but the distance of the new from the old label in the lattice, the amount of other consequence's labels changed, or the amount of affected users. We will also look at resolution of semantic conflicts and study a more expressive goal language to define for each single goal of a goal set whether it may be lowered or lifted in case of conflicts.

References

1. Baader, F., Knechtel, M., Peñaloza, R.: A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 49–64. Springer, Heidelberg (2009)
2. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. Journal of Logic and Computation 20(1), 5–34 (2010); Special Issue: Tableaux and Analytic Proof Methods
3. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: View-based query answering over description logic ontologies. In: Proc. of KR 2008 (2008)
4. Chen, W., Stuckenschmidt, H.: A model-driven approach to enable access control for ontologies. In: Proc. of WI 2009, pp. 663–672 (2009)
5. Farkas, C., Jajodia, S.: The inference problem: a survey. SIGKDD Explor. Newsl. 4(2), 6–11 (2002)
6. Gaag, A., Kohn, A., Lindemann, U.: Function-based solution retrieval and semantic search in mechanical engineering. In: Proc. of ICED 09 (2009)
7. Grau, B.C., Horrocks, I.: Privacy-preserving query answering in logic-based information systems. In: Proc. of ECAI 2008 (2008)
8. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 267–280. Springer, Heidelberg (2007)
9. Knechtel, M., Peñaloza, R.: Correcting access restrictions to a consequence. In: Proc. of DL 2010, CEUR-WS, vol. 573 (2010)
10. Knechtel, M., Peñaloza, R.: A generic approach for correcting access restrictions to a consequence. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 167–182. Springer, Heidelberg (2010)
11. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
12. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL queries for OWL-DL. In: Proc. of OWLED 2007 (2007)

On the Semantic Relationship between Datalog and Description Logics

Markus Krötzsch[1], Sebastian Rudolph[2], and Peter H. Schmitt[3]

[1] Oxford University Computing Laboratory, UK
markus.kroetzsch@comlab.ox.ac.uk
[2] Institute AIFB, Karlsruhe Institute of Technology, DE
sebastian.rudolph@kit.edu
[3] Institute for Theoretical Computer Science, Karlsruhe Institute of Technology, DE
pschmitt@ira.uka.de

Abstract. Translations to (first-order) datalog have been used in a number of inferencing techniques for description logics (DLs), yet the relationship between the semantic expressivities of function-free Horn logic and DL is understood only poorly. Although Description Logic Programs (DLP) have been described as DLs in the "expressive intersection" of DL and datalog, it is unclear what an intersection of two syntactically incomparable logics is, even if both have a first-order logic semantics. In this work, we offer a characterisation for DL fragments that can be expressed, in a concrete sense, in datalog. We then determine the largest such fragment for the DL $\mathcal{ALC}$, and provide an outlook on the extension of our methods to more expressive DLs.

1 Introduction

Ontologies and rules are two fundamental concepts in knowledge representation. Taking ontologies as the basic modelling paradigm has led to the development of Description Logics (DLs) with a wide range of successful knowledge representations languages. On the other hand rules are the central notion in Logic Programming building on first-order Horn logic. Both have been very prolific research areas and have recently received a boost in the context of the Semantic Web. As references for the purposes of this paper we point to [2] and [4]. Since decidability is an important concern for DL, function-free first-order Horn logic "*datalog*" is of particular interest.

Since the semantic frameworks for DL and datalog are very close it is natural that the research community started investigating the relationship between them. One direction explores how either formalism could be extended with features of the other. This line of research is represented by approaches such as $\mathcal{AL}$-log [8], CARIN [23], SWRL [13,14], $\mathcal{DL}$*+log* [28], DL-safe rules [27], DL Rules [21,11], but also Datalog$^{\pm}$ [5], and $\forall\exists$-rules [3]. Another direction aims at pin-pointing how both formalisms overlap. This has led to the study of Horn description logics [15,20] and Description Logic Programs (DLP) [12,29]. The latter is a family of DLs that can be faithfully expressed in first-order Horn-logic, and in particular in datalog, and the generalisation of this approach is the main topic of this paper.[1]

It is known that fragments of various DLs can be translated into equivalent or equisatisfiable datalog programs, and this has also been exploited to solve reasoning tasks. This has been demonstrated, e.g., for the description logics Horn-$\mathcal{SHIQ}$ and $\mathcal{EL}^{++}$ [15,22,17,18]. In this paper we address the question whether there is a maximal fragment that can be mapped into datalog. This would give a precise meaning to the slogan of the "expressive intersection" of DL and datalog. The failure of naive attempts to define maximal fragments eventually led to the definition of a DLP fragment for a given DL in Section 3 below. In Section 4 we define the DLP fragment $\mathcal{DLP}_{\mathcal{ALC}}$ of $\mathcal{ALC}$ and prove its maximality. This result can be extended to $\mathcal{SROIQ}$ but the necessary canonical syntactic descriptions are too complex to be included in this paper. We thus rather provide a summary of the relevant results and methods in Section 5 and refer for details and omitted proofs to the technical report [19].

2 Preliminaries

We assume the reader to be familiar with DLs (see [19,2] for details and references), and restrict to notational remarks here. The largest DL we encounter is $\mathcal{SROIQ}^{\text{free}}$, the well-known DL $\mathcal{SROIQ}$ without any restrictions on *simplicity* and *regularity* of roles, though only the simpler DL $\mathcal{ALC}$ will be considered in detail within this paper. DL knowledge bases are defined over finite sets of individual names (constants) $\mathbf{I}$, concept names $\mathbf{A}$, and roles $\mathbf{R}$. We call $\mathscr{S} = \langle \mathbf{I}, \mathbf{A}, \mathbf{R} \rangle$ a *signature*. A signature $\mathscr{S}' = \langle \mathbf{I}', \mathbf{A}', \mathbf{R}' \rangle$ is called an *extension* of $\mathscr{S}$, in symbols $\mathscr{S} \subseteq \mathscr{S}'$, if $\mathbf{I} \subseteq \mathbf{I}'$ and $\mathbf{A} \subseteq \mathbf{A}'$ and $\mathbf{R} \subseteq \mathbf{R}'$.

We use $\mathbf{FOL}_=$ to refer to standard first-order logic with equality. It is well known in the folklore of DL and easy to see that there exists a translation π of $\mathcal{SROIQ}$ and thus also of $\mathcal{ALC}$ into $\mathbf{FOL}_=$ that preserves logical inference, i.e. $\mathrm{KB}_1 \models \mathrm{KB}_2$ implies $\pi(\mathrm{KB}_1) \models \pi(\mathrm{KB}_2)$. A definition of π may e.g., be found in [19, Figure 3.4].

We use the term "*datalog*" to refer to the function-free Horn logic fragment of $\mathbf{FOL}_=$. A *datalog program* is a first-order theory which contains only formulae of the form $\forall \mathbf{x}.A_1 \wedge \ldots \wedge A_n \to B$ where A_i, B are atoms without function symbols of arity greater than 0, and universal quantifies over all variables occurring in the implications. We generally omit the quantifier, we simply write B if $n = 0$, and we use $\bot$ to denote the empty head.

It will not be sufficient for our work to consider knowledge bases KB such that $\pi(\mathrm{KB})$ is equivalent to a datalog program. Semantic equivalence turns out to be too restrictive, it does e.g., not allow the use of new constant symbols denoting individuals whose existence is required by ABox axioms. Equisatisfiability on the other hand is too weak – it does not preserve relevant logical entailments. The following notion turns out to be a more appropriate middle-ground:

Definition 1. *Given* $\mathbf{FOL}_=$ *theories* T *and* T' *with signatures* $\mathscr{S} \subseteq \mathscr{S}'$, *then* T' semantically emulates T *if*

[1] Besides these two strands on integrating first-order rules with DLs, there are numerous works on extending DLs with non-monotonic features from logic programming [10,9,28,25,26] which are interesting in their own right but not closely related to this work.

(1) every model of T' becomes a model of T when restricted to the interpretations of symbols from $\mathscr{S}$, and
(2) for every model $\mathcal{J}$ of T there is a model $\mathcal{I}$ of T' that has the same domain as $\mathcal{J}$, and that agrees with $\mathcal{J}$ on $\mathscr{S}$.

It is usually not necessary to mention the signatures of T and T' explicitly, since it is always possible to find minimal signatures for T and T' that satisfy condition (1) of Definition 1. The concept of semantic emulation is also known by the name *semantic conservative extension*, see e.g. [24, Def.11.29]. We will prefer semantic emulation for its brevity.

Definition 2. *Given* **FOL**$_=$ *theories T and T' with signatures $\mathscr{S} \subseteq \mathscr{S}'$, then T'* syntactically emulates *T if for every first-order formula φ over $\mathscr{S}$: $T \models \varphi$ iff $T' \models \varphi$.*

It is easy to see that *semantic emulation* implies *syntactic emulation*. This illustrates the strength and significance of *semantic emulation* for knowledge representation: whenever a theory T' semantically emulates a theory T, we find that T' and T encode the same information *about the symbols* in T, and in particular that T' cannot be distinguished from T when restricting to those symbols.

Note, *syntactic emulation* of T by T' can equivalently be characterized by the requirement that for every formula φ over $\mathscr{S}$ the sets $T \cup \{\varphi\}$ and $T' \cup \{\varphi\}$ be equisatisfiable.

We will later make use of the following lemma, which generalises the well-known least model property of datalog. The proof of this is straightforward by unravelling of the definitions.

Lemma 1. *Let $\mathcal{I}_1$, $\mathcal{I}_2$ be interpretations over the same domain which agree on the interpretation of constant and function symbols, and let T be a first-order theory that is satisfied by $\mathcal{I}_1$ and $\mathcal{I}_2$.*

1. *If T is a datalog program then also the intersection $\mathcal{I}_1 \cap \mathcal{I}_2$ satisfies T.*
2. *If T can be semantically emulated by a datalog program then also the intersection $\mathcal{I}_1 \cap \mathcal{I}_2$ satisfies T.*

The intersection of interpretations is defined in the obvious way based on the intersection of predicate extensions.

3 Considerations for Defining DLP

In this section, we discuss and motivate a generic definition for DLP fragments of a description logic. A powerful tool for obtaining this definition is the construction of variants of logical expressions which preserve only the logical structure but may modify concrete signature symbols:

Definition 3. *Let F be a* **FOL**$_=$ *formula, a DL axiom, or a DL concept expression, and let $\mathscr{S}$ be a signature. An expression F' is a* variant *of F in $\mathscr{S}$ if F' can be obtained from F by replacing each occurrence of a role/concept/individual name with some role/concept/individual name in $\mathscr{S}$. Multiple occurrences of the same entity name in F need* not *be replaced by the same entity name of $\mathscr{S}$ in this process.*

A knowledge base KB$'$ *is a* variant *of a knowledge base* KB *if it is obtained from* KB *by replacing each axiom with a variant.*

Note that we do not require all occurrences of an entity name to be renamed together, so it is indeed possible to obtain $A \sqcap \neg B$ from $A \sqcap \neg A$. Considering all variants of a formula or axiom allows us to study the semantics and expressivity of formulae based on their syntactic structure, disregarding any possible interactions between signature symbols. We therefore call a $\mathbf{FOL}_=$ formula, DL axiom, or DL concept expression F *name-separated* if no signature symbol occurs more than once in F.

Definition 4. *Given description logics $\mathcal{L}$ and $\mathcal{D}$, we call $\mathcal{D}$ a* DLP fragment *of $\mathcal{L}$ if*

(1) every axiom of $\mathcal{D}$ is an axiom of $\mathcal{L}$,
(2) there is a transformation function $\mathsf{datalog}$ *that maps a $\mathcal{D}$ axiom α to a datalog program* $\mathsf{datalog}(\alpha)$ *such that* $\mathsf{datalog}(\alpha)$ *semantically emulates α,*
(3) $\mathcal{D}$ is closed under variants, i.e. given any axiom α and an arbitrary variant α' of α, we find α is in $\mathcal{D}$ iff α' is.

Item (1) of this definition fixes the syntactic framework for DLP fragments. Item (2) states the property that motivates the study of DLP languages: every axiom of a DLP fragment can be expressed in datalog. DLP languages as discussed in the literature may require the use of auxiliary symbols for the translation to datalog [29], and the datalog program can no longer be semantically equivalent to the original knowledge base in this case, even if all consequences with respect to the original predicates are still the same. This motivates the use of semantic emulation as introduced in Definition 1.

Item (3) of Definition 4 reflects our desire to obtain fragments that correspond to well-behaved logical languages as opposed to being arbitrary collections of axioms. An obvious way to implement this would be to require DLP fragments to be described by a context-free grammar. A typical feature of grammars for logical languages is that they are parametrised by a logical signature that can be modified without changing the essential structural features of the language. This effect is mirrored by the requirement of item (3) without introducing detailed requirements on a suitable logical grammar. We will find grammatical descriptions in the cases we consider, though item (3) as such does not imply that this is possible.

Let us discuss for a moment an alternative to item (3) in Definition 4. It seems natural to require that membership in a fragment can be decided efficiently, say in polynomial time. Proposition 1 shows that in this case no maximal fragment can exist. Definition 4 allows fragments without any restriction on the complexity of the membership relation, but the maximal DLP fragment of $\mathcal{ALC}$ in Section 4 is described by a context-free language, and thus efficiently recognisable.

Proposition 1. *Given description logics $\mathcal{L}$ and $\mathcal{D}$, we call $\mathcal{D}$ a* P-DLP fragment *of $\mathcal{L}$ if items (1) and (2) of Definition 4 are satisfied, and in addition there is a polynomial procedure for deciding $\alpha \in \mathcal{D}$ for any DL axiom α.*

Unless the complexity classes P *and* PSPACE *coincide, there is no maximal* P-*DLP fragment of $\mathcal{ALC}$: given any* P-*DLP fragment $\mathcal{D}$ of $\mathcal{ALC}$, there is a* P-*DLP fragment $\mathcal{D}'$ of $\mathcal{ALC}$ that covers more axioms, i.e. $\mathcal{D} \subset \mathcal{D}'$.*

Proof. We start with an auxiliary construction: if the concept expression C is satisfiable and does not contain the symbols R, A_1, A_2, and c, then no datalog program semantically emulates the expression $\alpha_C := (C \sqcap \exists R.(A_1 \sqcup A_2))(c)$. For a contradiction, suppose that α_C is semantically emulated by a datalog theory $\mathsf{datalog}(\alpha_C)$. By construction, α_C is satisfiable, and so is $\{\alpha_C, A_i \sqsubseteq \bot\}$ for each $i = 1, 2$. By Definition 2, we find that $\mathsf{datalog}(\alpha_C) \cup \{A_i \sqsubseteq \bot\}$ is satisfiable, too. Thus, there are models $\mathcal{I}_i$ of $\mathsf{datalog}(\alpha_C)$ such that $A_i^{\mathcal{I}_i} = \emptyset$. By the least model property of datalog, there is also a model $\mathcal{I}$ of $\mathsf{datalog}(\alpha_C)$ such that $A_1^{\mathcal{I}} = A_2^{\mathcal{I}} = \emptyset$. But then $\mathsf{datalog}(\alpha_C) \cup \{A_1 \sqcup A_2 \sqsubseteq \bot\}$ is satisfiable although $\{\alpha, A_1 \sqcup A_2 \sqsubseteq \bot\}$ is not, contradicting the supposed semantic emulation.

Let us now assume for the sake of a contradiction that $\mathcal{D}$ contains all unsatisfiable $\mathcal{ALC}$ axioms of the form of α_C. This would give a polynomial decision procedure for deciding satisfiability of $\mathcal{ALC}$ concept expressions C: construct α_C from C (clearly polynomial) decide $\alpha_C \in \mathcal{D}$ (was assumed to be of polynomial complexity). This contradicts the fact that deciding (un)satisfiability of $\mathcal{ALC}$ concept expressions is PSpace hard.

Therefore, there is an unsatisfiable expression α with $\alpha \notin \mathcal{D}$. Now let $\mathcal{D}'$ be defined as $\mathcal{D} \cup \{\alpha\}$. The transformation is given by $\mathsf{datalog}'(\alpha) = \mathsf{datalog}(\alpha)$ if $\alpha \in \mathcal{D}$, and $\mathsf{datalog}'(\alpha) = \{\top \to A(x), A(x) \to \bot\}$ otherwise, where A is a new predicate symbol. It is immediate that $\mathcal{D}'$ P-DLP fragment of $\mathcal{ALC}$ strictly greater than $\mathcal{D}$. □

This proof exemplifies a general problem that occurs when trying to define DLP: the question whether an axiom is expressible in datalog is typically computationally harder than one would like to admit for a language definition. This result carries over to more expressive DLs, and remains valid even if requirements such as closure under common normal form transformations are added to the definition of fragments. The fact that this problem is avoided by item (3) in Definition 4 confirms our intuition that this requirement closely relates to the possibility of representing DLP fragments syntactically, i.e. without referring to complex semantic conditions.

Proposition 2. *Consider a class K of knowledge bases that belong to a DLP fragment of some description logic, and such that the maximal size of axioms in K is bounded. Deciding satisfiability of knowledge bases in K is possible in polynomial time.*

Proof. Let the maximal size of axioms be bounded by N. Let V be a vocabulary with N concept, role and constant symbols. By assumption we know that for every of the finitely many axioms α of size less than N there is a translation *datalog*(α). We use this as a (finite) look-up table in the definition of $\mathsf{datalog}_K(\beta)$ for axioms β in KB $\in K$: Find a renaming $\alpha = \sigma(\beta)$ such that α is an expression in the vocabulary V. Here σ is a usual 1-1 renaming of symbols, not a variant in the sense of Definition 3. Look up the datalog program $\mathsf{datalog}(\alpha)$ and set $\mathsf{datalog}_{\mathrm{KB}}(\beta) = \sigma(\mathsf{datalog}(\alpha))$. It is easy to see that $\mathsf{datalog}_{\mathrm{KB}}(\beta)$ still satisfies item (2) of Definition 4. Thus satisfiability of KB $\in K$ can be decided by checking satisfiability of $\bigcup_{\beta \in \mathrm{KB}} \mathsf{datalog}_{\mathrm{KB}}(\beta)$. The maximal number of variables occurring within these datalog programs may also be bounded by N. Satisfiability of datalog with at most N variables per rule can be decided in time polynomial in 2^N [7]. The renamings σ can likewise be found in time polynomial in 2^N. Since N is a constant, this yields a polynomial time upper bound for deciding satisfiability of knowledge bases in K. □

It is interesting that the previous result does not require any assumptions on the computational complexity of recognising or translating DLP axioms. Intuitively, Proposition 2

Concepts necessarily equivalent to $\top$:	$\mathbf{L}^{\mathcal{A}}_{\top} ::= \top \mid \forall\mathbf{R}.\mathbf{L}^{\mathcal{A}}_{\top} \mid \mathbf{L}^{\mathcal{A}}_{\top} \sqcap \mathbf{L}^{\mathcal{A}}_{\top} \mid \mathbf{L}^{\mathcal{A}}_{\top} \sqcup \mathbf{C}$
Concepts necessarily equivalent to $\bot$:	$\mathbf{L}^{\mathcal{A}}_{\bot} ::= \bot \mid \exists\mathbf{R}.\mathbf{L}^{\mathcal{A}}_{\bot} \mid \mathbf{L}^{\mathcal{A}}_{\bot} \sqcap \mathbf{C} \mid \mathbf{L}^{\mathcal{A}}_{\bot} \sqcup \mathbf{L}^{\mathcal{A}}_{\bot}$
Body ($C \in \mathbf{L}^{\mathcal{A}}_{B}$ iff $\neg C \sqsubseteq A$ in $\mathcal{DLP}_{\mathcal{ALC}}$):	$\mathbf{L}^{\mathcal{A}}_{B} ::= \mathbf{L}^{\mathcal{A}}_{\top} \mid \mathbf{L}^{\mathcal{A}}_{\bot} \mid \neg\mathbf{A} \mid \forall\mathbf{R}.\mathbf{L}^{\mathcal{A}}_{B} \mid \mathbf{L}^{\mathcal{A}}_{B} \sqcap \mathbf{L}^{\mathcal{A}}_{B} \mid \mathbf{L}^{\mathcal{A}}_{B} \sqcup \mathbf{L}^{\mathcal{A}}_{B}$
Head ($C \in \mathbf{L}^{\mathcal{A}}_{H}$ iff $A \sqsubseteq C$ in $\mathcal{DLP}_{\mathcal{ALC}}$):	$\mathbf{L}^{\mathcal{A}}_{H} ::= \mathbf{L}^{\mathcal{A}}_{B} \mid \mathbf{A} \mid \forall\mathbf{R}.\mathbf{L}^{\mathcal{A}}_{H} \mid \mathbf{L}^{\mathcal{A}}_{H} \sqcap \mathbf{L}^{\mathcal{A}}_{H} \mid \mathbf{L}^{\mathcal{A}}_{H} \sqcup \mathbf{L}^{\mathcal{A}}_{B}$
Assertions ($C \in \mathbf{L}^{\mathcal{A}}_{a}$ iff $C(a)$ in $\mathcal{DLP}_{\mathcal{ALC}}$):	$\mathbf{L}^{\mathcal{A}}_{a} ::= \mathbf{L}^{\mathcal{A}}_{H} \mid \exists\mathbf{R}.\mathbf{L}^{\mathcal{A}}_{a} \mid \mathbf{L}^{\mathcal{A}}_{a} \sqcap \mathbf{L}^{\mathcal{A}}_{a} \mid \mathbf{L}^{\mathcal{A}}_{a} \sqcup \mathbf{L}^{\mathcal{A}}_{B}$

Fig. 1. $\mathcal{DLP}_{\mathcal{ALC}}$ concepts in negation normal form

states that reasoning in any DLP language is necessarily "almost" tractable. Indeed, many DLs allow complex axioms to be decomposed into a number of simpler normal forms of bounded size, and in any such case tractability is obtained. Moreover, Proposition 2 clarifies why Horn-$\mathcal{SHIQ}$ cannot be in DLP: ExpTime worst-case complexity of reasoning can be proven for a class K of Horn-$\mathcal{SHIQ}$ knowledge bases as in the above proposition (see [20], noting that remaining complex axioms can be decomposed in Horn-$\mathcal{SHIQ}$).

4 The DLP Fragment of $\mathcal{ALC}$

Using Definition 4, it is now possible to investigate DLP fragments of relevant description logics. In this paper, we detail this approach for $\mathcal{ALC}$; some remarks on the more complex case of $\mathcal{SROIQ}$ are given in Section 5 below. It turns out that the largest DLP fragment of $\mathcal{ALC}$ exists, and can be defined as follows, where we use the negation normal form NNF for simplifying our presentation.

Definition 5. *We define the description logic $\mathcal{DLP}_{\mathcal{ALC}}$ to contain all knowledge bases consisting only of $\mathcal{ALC}$ axioms which are*

– *GCIs $C \sqsubseteq D$ such that* NNF$(\neg C \sqcup D)$ *is an $\mathbf{L}^{\mathcal{A}}_{H}$ concept as defined in Fig. 1, or*
– *ABox axioms $C(a)$ where* NNF(C) *is an $\mathbf{L}^{\mathcal{A}}_{a}$ concept as defined in Fig. 1.*

The headings in Fig. 1 give the basic intuition about the significance of the various concept languages. The distinction of head and body concepts is typical for many works on DLP and Horn DLs, while our use of additional assertional concepts takes into account that emulation allows for some forms of Skolemisation. Typical example representatives of the respective grammars are $\neg A \sqcap \forall R.(\neg B \sqcup \neg C)$ for $\mathbf{L}^{\mathcal{A}}_{B}$, $\neg A \sqcup (B \sqcap \forall R.C)$ for $\mathbf{L}^{\mathcal{A}}_{H}$, and $\neg A \sqcup \exists R.B$ for $\mathbf{L}^{\mathcal{A}}_{a}$.

Though name separation prevents most forms of semantic interactions within concepts, we still require grammars for $\mathbf{L}^{\mathcal{A}}_{\top}$ and $\mathbf{L}^{\mathcal{A}}_{\bot}$ to characterise concepts all variants of which are equivalent to $\top$ and $\bot$, respectively. This includes concept expressions such as $A \sqcap \exists R.\bot$ and $B \sqcup \forall R.\top$.

We start with an easy observation on Definition 5. This result will not explicitly be used later on but might add to the understanding of this definition.

Lemma 2. *Consider arbitrary $\mathcal{ALC}$ concept expressions C that do not contain quantifiers $\forall$, $\exists$, and the symbols $\top$ and $\bot$.*

1. *If $C \in \mathbf{L}_B^{\mathcal{A}}$ then C has a conjunctive normal form $\bigsqcap_i \bigsqcup_j C_{i,j}$ with $C_{i,j}$ a negated atom for all i, j.*
2. *If $C \in \mathbf{L}_H^{\mathcal{A}}$ or $C \in \mathbf{L}_a^{\mathcal{A}}$ then C has a conjunctive normal form $\bigsqcap_i \bigsqcup_j C_{i,j}$ with $C_{i,j}$ negated or unnegated atoms and for every i there is at most one j such that $C_{i,j}$ is an unnegated atom.*
   (Since the assumptions require that C does not contain quantifiers there is no difference here between $C \in \mathbf{L}_H^{\mathcal{A}}$ and $C \in \mathbf{L}_a^{\mathcal{A}}$.)

Proof. Notice, that $C \notin \mathbf{L}_\top^{\mathcal{A}}$ and $C \notin \mathbf{L}_\bot^{\mathcal{A}}$ since neither $\top$ nor $\bot$ occur in C. For item (1), note that if $C \in \mathbf{L}_B^{\mathcal{A}}$ then either C is a negated atom, or $C = C_1 \sqcap C_2$ or $C = C_1 \sqcup C_2$ with $C_i \in \mathbf{L}_B^{\mathcal{A}}$. The claim now follows easily from the induction hypothesis on C_1, C_2.

For item (2), by the assumptions on C we have $C \in \mathbf{L}_H^{\mathcal{A}}$ if one of the following cases holds true:

1. $C \in \mathbf{L}_B^{\mathcal{A}}$. Then the claim follows from part (1) of the lemma.
2. C is an atom. Then the claim is obviously true.
3. $C = C_1 \sqcap C_2$ with $C_i \in \mathbf{L}_H^{\mathcal{A}}$. If C_i' is a conjunctive normal form of C_i satisfying the claim then $C_1' \sqcap C_2'$ is a conjunctive normal form of C satisfying the claim.
4. $C = C_1 \sqcup C_2$ with $C_i \in \mathbf{L}_H^{\mathcal{A}}$ and $C_1 \in \mathbf{L}_B^{\mathcal{A}}$. Let $\bigsqcap_i \bigsqcup_j C_{i,j}^1$ and $\bigsqcap_m \bigsqcup_n C_{m,n}^2$ be the conjunctive normal forms that exist by induction hypothesis satisfying the respective claims. A conjunctive normal form of $C = C_1 \sqcup C_2$ is obtained as the conjunction of all $\bigsqcup_j C_{i,j}^1 \sqcup \bigsqcup_n C_{m,n}^2$ for all combinations of i, m. Since $\bigsqcup_j C_{i,j}^1$ contains at most one positive atom and $\bigsqcup_n C_{m,n}^2$ contains only negative atoms we are finished. □

It is obvious that $\mathcal{DLP}_{\mathcal{ALC}}$ satisfies items (1) and (3) of Definition 4, so what remains to show is that $\mathcal{DLP}_{\mathcal{ALC}}$ knowledge bases can indeed be expressed in datalog. Following the grammatical structure of $\mathcal{DLP}_{\mathcal{ALC}}$, we specify three auxiliary functions for constructing datalog programs to semantically emulate a $\mathcal{DLP}_{\mathcal{ALC}}$ knowledge base. The following two lemmata can be proven by simple inductions, see [19] for further details.

Lemma 3. *Given a concept name A, and a concept $C \in \mathbf{L}_H^{\mathcal{A}}$, Fig. 2 recursively defines a datalog program $\mathsf{dlg}_H^{\mathcal{A}}(A \sqsubseteq C)$ that semantically emulates $A \sqsubseteq C$.*

For an example of this transformation, consider the $\mathbf{L}_H^{\mathcal{A}}$ concept $E = \neg B \sqcup (C \sqcap \forall R.D)$. Then $\mathsf{dlg}_H^{\mathcal{A}}(A \sqsubseteq E)$ consists of the following rules:

$$\begin{aligned}
A(x) \land X_1(x) &\rightarrow X_2(x)\\
B(x) &\rightarrow X_1(x)\\
X_2(x) &\rightarrow C(x)\\
X_2(x) \land R(x,y) &\rightarrow X_3(x)\\
X_3(x) &\rightarrow D(x)
\end{aligned}$$

Clearly, this rule set could be further simplified to obtain the three rules $A(x) \land B(x) \rightarrow X_2(x)$, $X_2(x) \rightarrow C(x)$, $X_2(x) \land R(x,y) \rightarrow D(x)$ which are easily seen to semantically emulate $A \sqsubseteq E$.

Lemma 4. *Given a constant a and a concept $C \in \mathbf{L}_a^{\mathcal{A}}$, Fig. 3 recursively defines a datalog program $\mathsf{dlg}_H^{\mathcal{A}}(C(a), \bot)$ that semantically emulates $C(a)$.*

C	$\mathsf{dlg}^{\mathcal{A}}_{H}(A \sqsubseteq C)$
$D \in \mathbf{L}^{\mathcal{A}}_{B}$	$\mathsf{dlg}^{\mathcal{A}}_{B}(\neg X \sqsubseteq D) \cup \{A(x) \wedge X(x) \rightarrow \bot\}$
B	$\{A(x) \rightarrow B(x)\}$
$\forall R.D$	$\mathsf{dlg}^{\mathcal{A}}_{H}(X \sqsubseteq D)$ $\cup \{A(x) \wedge R(x,y) \rightarrow X(y)\}$
$D_1 \sqcap D_2$	$\mathsf{dlg}^{\mathcal{A}}_{H}(A \sqsubseteq D_1) \cup \mathsf{dlg}^{\mathcal{A}}_{H}(A \sqsubseteq D_2)$
$D_1 \sqcup D_2 \in (\mathbf{L}^{\mathcal{A}}_{H} \sqcup \mathbf{L}^{\mathcal{A}}_{B})$	$\mathsf{dlg}^{\mathcal{A}}_{H}(X_2 \sqsubseteq D_1) \cup \mathsf{dlg}^{\mathcal{A}}_{B}(\neg X_1 \sqsubseteq D_2) \cup \{A(x) \wedge X_1(x) \rightarrow X_2(x)\}$

C	$\mathsf{dlg}^{\mathcal{A}}_{B}(\neg A \sqsubseteq C)$
$D \in \mathbf{L}^{\mathcal{A}}_{\top}$	$\{\}$
$D \in \mathbf{L}^{\mathcal{A}}_{\bot}$	$\{A(x)\}$
$\neg B$	$\{B(x) \rightarrow A(x)\}$
$\forall R.D$	$\mathsf{dlg}^{\mathcal{A}}_{B}(\neg X \sqsubseteq D) \cup \{R(x,y) \wedge X(y) \rightarrow A(x)\}$
$D_1 \sqcap D_2 \in (\mathbf{L}^{\mathcal{A}}_{B} \sqcap \mathbf{L}^{\mathcal{A}}_{B})$	$\mathsf{dlg}^{\mathcal{A}}_{B}(\neg A \sqsubseteq D_1) \cup \mathsf{dlg}^{\mathcal{A}}_{B}(\neg A \sqsubseteq D_2)$
$D_1 \sqcup D_2 \in (\mathbf{L}^{\mathcal{A}}_{B} \sqcup \mathbf{L}^{\mathcal{A}}_{B})$	$\mathsf{dlg}^{\mathcal{A}}_{B}(\neg X_1 \sqsubseteq D_1) \cup \mathsf{dlg}^{\mathcal{A}}_{B}(\neg X_2 \sqsubseteq D_2) \cup \{X_1(x) \wedge X_2(x) \rightarrow A(x)\}$
A, B concept names, R a role, $X_{(i)}$ fresh concept names	

Fig. 2. Transforming axioms $\mathbf{A} \sqsubseteq \mathbf{L}^{\mathcal{A}}_{H}$ and $\neg\mathbf{A} \sqsubseteq \mathbf{L}^{\mathcal{A}}_{B}$ to datalog

Again, this transformation is designed for a concise definition, not for optimised output. For an example, consider the $\mathbf{L}^{\mathcal{A}}_{a}$ concept $E = \neg B \sqcup \exists R.C$. Then $\mathsf{dlg}^{\mathcal{A}}_{H}(E(a), \bot)$ consists of the following rules (X_i and Y indicating fresh concept names as in the definition of the transformation):

$$\begin{array}{rcl@{\qquad}rcl}
B(x) & \rightarrow & X_1(x) & X_2(a) & \rightarrow & R(a,b) \\
X_2(a) & \rightarrow & Y(b) & X_3(x) \wedge X_4(x) & \rightarrow & X_2(x) \\
 & \rightarrow & X_3(x) & X_1(x) & \rightarrow & X_4(x) \\
 & \rightarrow & X_5(b) & X_5(x) \wedge X_6(x) & \rightarrow & X_7(x) \\
X_7(x) & \rightarrow & C(x) & Y(x) & \rightarrow & X_6(x)
\end{array}$$

As before, this rule set can be simplified significantly by eliminating most of the introduced auxiliary concept symbols. Doing this, we obtain the three rules $B(x) \rightarrow X_2(x)$, $X_2(a) \rightarrow R(a,b)$, and $X_2(a) \rightarrow C(b)$, which again are easily seen to semantically emulate $E(a)$ as claimed. Here, the fresh constant symbol b acts as a Skolem constant that represents the individual that the existential concept expression may require to exist.

Combining the previous lemmata, we obtain the emulation theorem for $\mathcal{DLP}_{\mathcal{ALC}}$.

Theorem 1. *For every $\mathcal{DLP}_{\mathcal{ALC}}$ axiom α as in Definition 5, one can construct a datalog program* $\mathsf{dlg}(\alpha)$ *that emulates α.*

Proof. If $\alpha = C \sqsubseteq D$ is a TBox axiom, define $\mathsf{datalog}(\alpha) := \mathsf{dlg}^{\mathcal{A}}_{H}(A \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)) \cup \{A(x)\}$. If $\alpha = C(a)$ is an ABox axiom, define $\mathsf{datalog}(\alpha) := \mathsf{dlg}^{\mathcal{A}}_{a}(C(a), \bot)$. The result follows by Lemma 3 and 4. □

C	$\mathsf{dlg}_a^{\mathcal{A}}(C(a), E)$
$D \in \mathbf{L}_H^{\mathcal{A}}$	$\mathsf{dlg}_H^{\mathcal{A}}(X \sqsubseteq D \sqcup E) \cup \{X(a)\}$
$D_1 \sqcap D_2$	$\mathsf{dlg}_a^{\mathcal{A}}(D_1(a), E) \cup \mathsf{dlg}_a^{\mathcal{A}}(D_2(a), E)$
$D_1 \sqcup D_2 \in (\mathbf{L}_a^{\mathcal{A}} \sqcup \mathbf{L}_B^{\mathcal{A}})$	$\mathsf{dlg}_B^{\mathcal{A}}(\neg X \sqsubseteq D_2) \cup \mathsf{dlg}_a^{\mathcal{A}}(D_1(a), E \sqcup \neg X)$
$\exists R.D$	$\mathsf{dlg}_B^{\mathcal{A}}(\neg X \sqsubseteq E) \cup \mathsf{dlg}_a^{\mathcal{A}}(D(b), \neg Y) \cup \{X(a) \to R(a,b), X(a) \to Y(b)\}$
$E \in \mathbf{L}_B^{\mathcal{A}}$, X, Y fresh concept names, b a fresh constant	

Fig. 3. Transforming axioms $C(a)$ with $C \in \mathbf{L}_a^{\mathcal{A}}$ to datalog

We still need to show that $\mathcal{DLP}_{\mathcal{ALC}}$ is indeed the largest DLP fragment of $\mathcal{ALC}$. We first introduce two transformations – etb and qe –, and make some basic observations that allow us to use these transformations for showing maximality of $\mathcal{DLP}_{\mathcal{ALC}}$.

Definition 6. *Let C be an arbitrary $\mathcal{ALC}$ concept expression. The expression etb(C) (*eliminate **t**op and **b**ottom*) is obtained from C by elimination of top and bottom symbols, achieved by applying exhaustively the following rewrite rules:*

$$\begin{array}{lllll} \top \sqcap D \mapsto D & \bot \sqcup D \mapsto D & \top \sqcup D \mapsto \top & \bot \sqcap D \mapsto \bot & \forall \mathbf{R}.\top \mapsto \top \\ D \sqcap \top \mapsto D & D \sqcup \bot \mapsto D & D \sqcup \top \mapsto \top & D \sqcap \bot \mapsto \bot & \exists \mathbf{R}.\bot \mapsto \bot \end{array}$$

Note, that etb(C) may still contain subexpressions of the form $\forall \mathbf{R}.\bot$ and $\exists \mathbf{R}.\top$

The next lemma summarises some easy observations on etb.

Lemma 5. *For any $\mathcal{ALC}$ concept expression C*

1. *etb(C) is logically equivalent to C, i.e., for any interpretation $\langle \Delta^{\mathcal{I}}, \mathcal{I} \rangle$ and any $a \in \Delta^{\mathcal{I}}$, we have $a \in C^{\mathcal{I}}$ iff $a \in etb(C)^{\mathcal{I}}$.*
2. *$C \in \mathbf{L}_\top^{\mathcal{A}}$ iff $etb(C) \in \mathbf{L}_\top^{\mathcal{A}}$ $\quad C \in \mathbf{L}_\bot^{\mathcal{A}}$ iff $etb(C) \in \mathbf{L}_\bot^{\mathcal{A}}$ $\quad C \in \mathbf{L}_a^{\mathcal{A}}$ iff $etb(C) \in \mathbf{L}_a^{\mathcal{A}}$*
   $C \in \mathbf{L}_B^{\mathcal{A}}$ iff $etb(C) \in \mathbf{L}_B^{\mathcal{A}}$ $\quad C \in \mathbf{L}_H^{\mathcal{A}}$ iff $etb(C) \in \mathbf{L}_H^{\mathcal{A}}$
3. *If C does not contain subexpressions of the form $\forall \mathbf{R}.\bot$ or $\exists \mathbf{R}.\top$ then $etb(C) = \bot$, or $etb(C) = \top$, or etb(C) does neither contain $\bot$ nor $\top$.*

Definition 7. *Let C be an arbitrary $\mathcal{ALC}$ concept expression. The expression qe(C) is obtained from C by* quantifier elimination*:*

$$\begin{array}{llll} qe(A) & = A \quad \textit{(concept name)} & qe(\neg C_1) & = \neg\, qe(C_1) \\ qe(C_1 \sqcap C_2) & = qe(C_1) \sqcap qe(C_2) & qe(C_1 \sqcup C_2) & = qe(C_1) \sqcup qe(C_2) \\ qe(\forall R.C_1) & = qe(C_1) & qe(\exists R.C_1) & = qe(C_1) \end{array}$$

Lemma 6. *Let $\langle \mathbf{I}, \mathbf{A}, \mathbf{R} \rangle$ be a signature and fix a domain Δ. There is an interpretation $\mathcal{I}_1$ on Δ of the role symbols in $\mathbf{R}$ such that for any interpretation $\mathcal{I}_0$ on Δ of the signature $(\mathbf{I}, \mathbf{A}, \emptyset)$, and for any concept C of $\langle \mathbf{I}, \mathbf{A}, \mathbf{R} \rangle$, we find $C^{\mathcal{I}} = qe(C)^{\mathcal{I}_0}$ with $\mathcal{I} = \mathcal{I}_0 \cup \mathcal{I}_1$.*

Proof. Setting $\mathcal{I}_1(R) = \{\langle a, a \rangle \mid a \in \Delta\}$ for all $R \in \mathbf{R}$, we obtain:

$(\forall R.D)^{\mathcal{I}} = \{a \in \Delta \mid b \in D^{\mathcal{I}_0}$ for all $\langle a, b \rangle \in R^{\mathcal{I}_1}\} = \{a \in \Delta \mid a \in D^{\mathcal{I}_0}\} = D^{\mathcal{I}_0}$,
$(\exists R.D)^{\mathcal{I}} = \{a \in \Delta \mid$ there is $\langle a, b \rangle \in R^{\mathcal{I}_1}$ with $b \in D^{\mathcal{I}_0}\} = \{a \in \Delta \mid a \in D^{\mathcal{I}_0}\} = D^{\mathcal{I}_0}$. □

Note, that Lemma 6 is true for arbitrary $\mathcal{ALC}$ concept expressions, they need neither belong to $\mathcal{DLP}_{\mathcal{ALC}}$ nor be name-separated.

Lemma 7. *Let C be an arbitrary $\mathcal{ALC}$ concept expression. Then*

$$\begin{array}{lll} C \in \mathbf{L}_B^{\mathcal{A}} & \textit{iff} & qe(C) \in \mathbf{L}_B^{\mathcal{A}}, \\ C \in \mathbf{L}_H^{\mathcal{A}} & \textit{iff} & qe(C) \in \mathbf{L}_H^{\mathcal{A}}, \\ C \in \mathbf{L}_a^{\mathcal{A}} & \textit{iff} & qe(C) \in \mathbf{L}_a^{\mathcal{A}}. \end{array}$$

Proof. Here is a sample from the inductive proof for the first equivalence. The goal in this case is to show that $(\forall R.D) \in \mathbf{L}_B^{\mathcal{A}}$ iff $D \in \mathbf{L}_B^{\mathcal{A}}$.

The "if" direction is directly covered by a grammar rule. For the "only if" direction, we observe that there are only two grammar rules that can produce a formula of the form $(\forall R.D)$. The first is $\forall \mathbf{R}.\mathbf{L}_B^{\mathcal{A}}$, for which we directly find that $(\forall R.D) \in \mathbf{L}_B^{\mathcal{A}}$ implies $D \in \mathbf{L}_B^{\mathcal{A}}$. The second rule is $\forall R.\mathbf{L}_\top^{\mathcal{A}}$. Thus $(\forall R.D) \in \mathbf{L}_B^{\mathcal{A}}$ implies $D \in \mathbf{L}_\top^{\mathcal{A}}$, which suffices since $\mathbf{L}_\top^{\mathcal{A}} \subseteq \mathbf{L}_B^{\mathcal{A}}$. □

Theorem 2. *$\mathcal{DLP}_{\mathcal{ALC}}$ is the largest DLP fragment of $\mathcal{ALC}$.*

Proof. For a contradiction, suppose that there is a DLP fragment $\mathcal{F}$ of $\mathcal{ALC}$ that is strictly larger than $\mathcal{DLP}_{\mathcal{ALC}}$. Then there is some GCI $C' \sqsubseteq D'$ in $\mathcal{F}$ but not in $\mathcal{DLP}_{\mathcal{ALC}}$. The other possibility that there is an ABox axiom $C'(a) \in \mathcal{F}$ with $C'(a) \notin \mathbf{L}_a^{\mathcal{A}}$ is completely analogous. By Definition 4, any name-separated variant $C \sqsubseteq D$ of $C' \sqsubseteq D'$ is still in $\mathcal{F}$. Since $\mathcal{DLP}_{\mathcal{ALC}}$ is closed under variants, $C \sqsubseteq D$ is not in $\mathcal{DLP}_{\mathcal{ALC}}$. By Definition 5 this means that the negation normal form E of $\neg C \sqcup D$ is not in $\mathbf{L}_H^{\mathcal{A}}$. By Lemmas 5 and 7 also etb(qe(E)) is not in $\mathbf{L}_H^{\mathcal{A}}$. Let E^{cnf} be a conjunctive normal form of etb(qe(E)). Thus $E^{cnf} = Con_1 \sqcap \ldots \sqcap Con_k$ with $Con_i = L_{i,1} \sqcup \ldots \sqcup L_{i,n_i}$ where each $L_{i,j}$ is a concept name or the negation of a concept name. Again, it can be verified that $E \in \mathbf{L}_H^{\mathcal{A}}$ iff $E^{cnf} \in \mathbf{L}_H^{\mathcal{A}}$. Furthermore, for one i, $1 \leq i \leq k$ there are two unnegated concept names among $\{L_{i,1}, \ldots, L_{i,n_i}\}$. Otherwise, we could show $E^{cnf} \in \mathbf{L}_H^{\mathcal{A}}$. For this we need the extended grammar of $\mathbf{L}_H^{\mathcal{A}}$. Without loss of generality let $i = 1$ and $L_{1,1} = A_1$, $L_{1,2} = A_2$ positive. The name separation of E may have been lost by building the transformation to conjunctive normal form E^{cnf}, but we still have the following:

1. For any atom A, if A occurs in E^{cnf} then $\neg A$ does not occur in E^{cnf}, and vice versa.
2. For any two different conjuncts Con_i and Con_j of E^{cnf}, there is a literal l occurring in Con_i and not in Con_j (and by symmetry also a literal l' occurring in Con_j and not in Con_i).

Claim 1 can be easily seen since the transformation from E to E^{cnf} is effected by repeated application of the rewriting rule $(C_1 \sqcap C_2) \sqcup C_3 \mapsto (C_1 \sqcup C_3) \sqcap (C_2 \sqcup C_3)$.

Claim 2 can be proven by induction on the structural complexity of E. In the simplest case E already is a conjunctive normal from. Then name separation of E even implies that different conjuncts Con_i are disjoint. Next assume that $E = E_1 \sqcup \ldots \sqcup E_n$ and by induction hypothesis each E_i has a conjunctive normal form $E_i = Con_{i,1} \sqcap \ldots \sqcap Con_{i,m_i}$, such that for $j \neq k$ the conjunct $Con_{i,j}$ contains a literal, that does not occur in $Con_{i,k}$.

Furthermore, name separation of E tells us that different E_{i_1}, E_{i_2} do not share a literal. By elementary computation we have

$$E^{cnf} = \bigsqcap_{1 \le i_1 \le m_1} \dots \bigsqcap_{1 \le i_n \le m_n} (Con_{1,i_1} \sqcup \dots \sqcup Con_{n,i_n})$$

Let us look at two different conjuncts in E^{cnf}. Typically we may consider $Con_{1,1} \sqcup C_r$ and $Con_{1,2} \sqcup C_r$ with $C_r = Con_{2,i_2} \sqcup \dots \sqcup Con_{n,i_n}$. By induction hypothesis there is a literal l in $Con_{1,1}$ that does not occur in $Con_{1,2}$. Under the present assumptions l occurs in $Con_{1,1} \sqcup C_r$ and not in $Con_{1,2} \sqcup C_r$. This completes our proof of claim 2. Returning to our main line of reasoning we define interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ on a universe Δ by

$$\begin{array}{ll} A_1^{\mathcal{I}_1} = \Delta & L_{1,j}^{\mathcal{I}_1} = \emptyset \text{ for all } 2 \le j \le n_1 \\ A_2^{\mathcal{I}_2} = \Delta & L_{1,j}^{\mathcal{I}_2} = \emptyset \text{ for all } 1 \le j \le n_1, j \ne 2 \end{array}$$

Thus

$$Con_1^{\mathcal{I}_1} = Con_1^{\mathcal{I}_2} = \Delta \quad \text{and} \quad Con_1^{\mathcal{I}_1 \cap \mathcal{I}_2} = \emptyset$$

By property 2 it is possible to extend the interpretations $\mathcal{I}_i$ such that $Con_j^{\mathcal{I}_i} = \Delta$ for $i \in \{i, 1\}$ and $2 \le j \le k$. In total we have

$$(E^{cnf})^{\mathcal{I}_1} = (E^{cnf})^{\mathcal{I}_2} = \Delta \quad \text{and} \quad (E^{cnf})^{\mathcal{I}_1 \cap \mathcal{I}_2} = \emptyset$$

Since the normal form and the etb transformation preserve logical equivalence, we also have $\text{qe}(E)^{\mathcal{I}_i} = \Delta$ for $i = 1, 2$ and $\text{qe}(E)^{\mathcal{I}_1 \cap \mathcal{I}_2} = \emptyset$. By Lemma 6 there are expansions $\mathcal{I}_i^*$ of $\mathcal{I}_i$ such that $E^{\mathcal{I}_i^*} = \text{qe}(E)^{\mathcal{I}_i} = \Delta$ for $i \in \{1, 2\}$ and $E^{\mathcal{I}_1{}^* \cap \mathcal{I}_2{}^*} = E^{(\mathcal{I}_1 \cap \mathcal{I}_2)^*} = \text{qe}(E)^{\mathcal{I}_1 \cap \mathcal{I}_2} = \emptyset$. By Lemma 1, this contradicts the possibility that $\pi(E)$ can be emulated by a datalog formula. □

5 The Datalog Fragment of $\mathcal{SROIQ}$

The previous section showed that syntactic descriptions tend to become rather complex when maximising languages in a canonical way, but the situation is substantially more intricate when considering $\mathcal{SROIQ}^{\text{free}}$ instead of $\mathcal{ALC}$ as an underlying DL. Here, we summarise the conclusions that have been obtained in [19] for this case. There, a maximal DLP fragment of $\mathcal{SROIQ}^{\text{free}}$ has been developed under the additional requirement of closure under disjunctive normal forms (DNF):

Theorem 3. *The largest DL fragment of $\mathcal{SROIQ}^{\text{free}}$ that is also closed under DNF exists, and it can be characterised by a parametrised set of grammar productions. We call this DL $\mathcal{DLP}$.*

Disjunctive normal forms here are mainly required to curtail the syntactic complexity of the obtained fragment, and we conjecture that a maximal DLP fragment of $\mathcal{SROIQ}^{\text{free}}$ that does not have this property also exists. Rather than in the concrete description of this fragment, we are interested here in the general insights that are obtained from proofs of such results. The above result consists of three parts: (1) specifying an explicit syntactic characterisation, (2) showing that all $\mathcal{DLP}$ axioms can be $\mathbf{FOL}_=$-emulated in datalog, (3) showing that $\mathcal{DLP}$ is the largest such DL. Here we give an overview of the main methods that are used in each step.

Syntactic Characterisation. The main challenge here is to reduce the presentational complexity as far as possible. A *DLP normal form* is introduced that incorporates DNF and an improved form of NNF, and which ignores concepts that, like $\mathbf{L}^{\mathcal{A}}_{\top}/\mathbf{L}^{\mathcal{A}}_{\bot}$ above, are always equivalent to $\top/\bot$. The syntax of $\mathcal{DLP}$ in normal form is still very complex due to the interplay of number restrictions and nominals that is possible even in name-separated axioms.

Datalog Emulation. A recursive datalog transformation as in the case of $\mathcal{DLP}_{\mathcal{ALC}}$ above is provided. The individual steps are substantially more involved, and even lead to exponentially large datalog programs in various cases, although these programs are very regular and can be constructed in a single pass without complex computations. We conjecture that this blow-up is unavoidable but this issue has not been investigated further.

Maximality. The least model property of datalog was used for showing maximality of $\mathcal{DLP}_{\mathcal{ALC}}$, but no extension of this direct approach to $\mathcal{DLP}$ has been found. Instead, additional model-theoretic properties of datalog were used that incorporate submodels and product models [6]. Using various inductive arguments, it has then been shown that any extension of $\mathcal{DLP}$ leads to axioms that cannot be $\mathbf{FOL}_=$-emulated in datalog.

We provide some examples to illustrate the issues that occur in the general case (datalog emulations are provided in parentheses). DLP expressions of the form $A \sqcap \exists R.B \sqsubseteq \forall S.C$ ($A(x) \wedge R(x,y) \wedge B(y) \wedge S(x,z) \to C(z)$) are well-known. The same is true for $A \sqsubseteq \exists R.\{c\}$ ($A(x) \to R(x,c)$) but hardly for $A \sqsubseteq {\geqslant}2\,R.(\{c\} \sqcup \{d\})$ ($A(x) \to R(x,c)$, $A(x) \to R(x,d)$, $A(x) \wedge c \approx d \to \bot$). Another unusual form of DLP axioms arises when Skolem constants (not functions) can be used as in the case $\{c\} \sqsubseteq {\geqslant}2\,R.A$ ($R(c,s)$, $R(c,s')$, $A(s)$, $A(s')$, $s \approx s' \to \bot$ with fresh s, s') and $A \sqsubseteq \exists R.(\{c\} \sqcap \exists S.\top)$ ($A(x) \to R(x,c)$, $A(x) \to S(c,s)$ with fresh s). This is possible since semantic emulation is more general than semantic equivalence.

For a more complex $\mathcal{DLP}$ axiom, consider the GCI $\{c\} \sqsubseteq {\geqslant}2\,R.(\neg\{a\} \sqcup A \sqcup B)$. It is semantically emulated by $\{R(c,s_1), R(c,s_2), a \approx s_1 \to A(s_1), a \approx s_2 \to A(s_2)\}$ where s_i are fresh constants. Note how equalities of fresh constants are used to simulate finite amounts of disjunctive behaviour. In contrast, $\{c\} \sqsubseteq {\geqslant}2\,R.(\neg\{a\} \sqcup A \sqcup B \sqcup C)$ is not in $\mathcal{DLP}$.

Another complex example is $\{c\} \sqsubseteq {\geqslant}4\,R.(A \sqcup \{a\} \sqcup (\{b\} \sqcap {\leqslant}1\,S.(\{c\} \sqcup \{d\})))$ which is semantically emulated by a datalog program that contains about 30 rules. Interestingly, the axiom $\{c\} \sqsubseteq {\geqslant}3\,R.(A \sqcup \{a\} \sqcup (\{b\} \sqcap {\leqslant}1\,S.(\{c\} \sqcup \{d\})))$ which only differs by using 3 instead of 4 cannot be $\mathbf{FOL}_=$-emulated by any datalog program.

6 Conclusions and Outlook

DLP provides an interesting example of a general type of problem: given two KR formalisms that can be translated to first-order logic, how can we syntactically characterise all theories of the source formalism that can faithfully be represented in the target formalism? In this work, we proposed to interpret "faithful representation" by means of semantic emulation (a weaker notion of semantic equivalence), while "syntactic" has been realised by requiring closure under variants (non-uniform renamings of signature symbols). These two simple principles allowed us to show the existence of a largest DLP fragment for the DL $\mathcal{ALC}$. In this sense, we argue that our approach introduces a workable definition for the vague notion of the "intersection" of two KR formalisms.

Our rigorous definition of DLP fragments also clarifies the differences between DLP and the DLs $\mathcal{EL}$ and Horn-$\mathcal{SHIQ}$ which can both be expressed in terms of datalog as well. Neither $\mathcal{EL}$ nor Horn-$\mathcal{SHIQ}$ can be semantically emulated in datalog but both satisfy a weaker version of syntactic emulation that is obtained by restricting to variable-free formulae φ in Definition 2. Under such weaker requirements, a larger space of possible DL fragments is allowed, but it is unknown whether (finitely many) maximal languages exist in this case. There is clearly no largest such language, since both $\mathcal{EL}$ and $\mathcal{DLP}$ abide by the weakened principles whereas their (intractable) union does not (contradicting Proposition 2).

Even when weakening the requirements of DLP fragments like this, Horn-$\mathcal{SHIQ}$ is still excluded by Proposition 2, which explains why Horn-$\mathcal{SHIQ}$ cannot be translated to datalog axiom-by-axiom. In the presence of transitivity, Horn-$\mathcal{SHIQ}$ also is not really closed under variants, but this problem could be overcome by using distinct signature sets for simple and non-simple roles. Again, it is open which results can be established for Horn-$\mathcal{SHIQ}$-like DLs based on the remaining weakened principles.

This work also explicitly introduces a notion of *emulation* which appears to be novel, though loosely related to conservative extensions. In essence, it requires that a theory can take the place of another theory in all logical contexts, based on a given syntactic interface. Examples given in this paper illustrate that this can be very different from semantic equivalence. Yet, emulation can be argued to define minimal requirements for preserving a theory's semantics even in combination with additional information, so it appears to be a natural tool for enabling information exchange in distributed knowledge systems. We think that the articulation of this notion is useful for studying the semantic interplay of heterogeneous logical formalisms in general.

Finally, the approach of this paper – seeking a logical fragment that is provably maximal under certain conditions – immediately leads to a number of further research questions. For example, what is the maximal fragment of SWRL ("datalog $\cup$ $\mathcal{SROIQ}$," see [14]) that can be expressed in $\mathcal{SROIQ}$? Clearly, this fragment would contain DL Rules [21] and maybe some form of DL-safe rules [27]. But also the maximal $\mathbf{FOL}_=$ fragment that can be expressed in a well-known subset such as the Guarded Fragment [1] or the two-variable fragment might be of general interest. We argue that ultimate answers to such questions can indeed be obtained based on similar definitions of *fragments* as used for DLP in this work. At the same time, our study of $\mathcal{SROIQ}$ indicates that the required definitions and arguments can become surprisingly complex when dealing with a syntactically rich formalism like description logic. The main reason for this is that constructs that are usually considered "syntactic sugar" have non-trivial semantic effects when considering logical fragments that are closed under variants.

Acknowledgements. The first author has been working at Karlsruhe Institute of Technology when conducting the research reported herein. This work was supported by DFG in project *ExpresST* and by EPSRC in project *ConDOR* (EP/G02085X/1).

References

1. Andréka, H., van Benthem, J.F.A.K., Németi, I.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic 27(3), 217–274 (1998)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications, 2nd edn. Cambridge University Press, Cambridge (2007)
3. Baget, J.F., Leclere, M., Mugnier, M.L.: Walking the decidability line for rules with existential variables. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10), pp. 466–476. AAAI Press, Menlo Park (2010)
4. Baral, C., Gelfond, M.: Logic programming and knowledge representation. J. Log. Program. 19(20), 73–148 (1994)
5. Calì, A., Gottlob, G., Lukasiewicz, T.: Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In: Fagin, R. (ed.) Proc. 12th Int. Conf. on Database Theory (ICDT'09), pp. 14–30. ACM, New York (2009)
6. Chang, C., Keisler, H.J.: Model Theory, 3rd edn. Studies in Logic and the Foundations of Mathematics, vol. 73. North-Holland, Amsterdam (1990)
7. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Computing Surveys 33(3), 374–425 (2001)
8. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: $\mathcal{AL}$-log: Integrating datalog and description logics. Journal of Intelligent and Cooperative Information Systems 10(3), 227–252 (1998)
9. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: Kaelbling, Saffiotti (eds.) [16], pp. 90–96
10. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. In: Dubois, D., Welty, C.A., Williams, M.A. (eds.) Proc. 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'04), pp. 141–151. AAAI Press, Menlo Park (2004)
11. Gasse, F., Sattler, U., Haarslev, V.: Rewriting rules into $\mathcal{SROIQ}$ axioms. In: Baader, F., Lutz, C., Motik, B. (eds.) Proc. 21st Int. Workshop on Description Logics (DL'08). CEUR Workshop Proceedings, vol. 353, CEUR-WS.org. (2008)
12. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proc. 12th Int. Conf. on World Wide Web (WWW'03), pp. 48–57. ACM, New York (2003)
13. Horrocks, I., Patel-Schneider, P.F.: A proposal for an OWL rules language. In: Feldman, S.I., Uretsky, M., Najork, M., Wills, C.E. (eds.) Proc. 13th Int. Conf. on World Wide Web (WWW'04), pp. 723–731. ACM, New York (2004)
14. Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., Tsarkov, D.: OWL Rules: A proposal and prototype implementation. Journal of Web Semantics 3(1), 23–40 (2005)
15. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Kaelbling, Saffiotti (eds.) [16], pp. 466–471
16. Kaelbling, L., Saffiotti, A. (eds.): Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). Professional Book Center (2005)

17. Kazakov, Y.: Saturation-Based Decision Procedures for Extensions of the Guarded Fragment. Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany (2006)
18. Krötzsch, M.: Efficient inferencing for OWL EL. In: Proc. 12th European Conf. on Logics in Artificial Intelligence (JELIA'10). LNCS (LNAI). Springer, Heidelberg (to appear, 2010)
19. Krötzsch, M., Rudolph, S.: Finding the largest datalog fragment of description logic. Tech. Rep. 3002, Institute AIFB, Karlsruhe Institute of Technology (2009), http://www.aifb.kit.edu/web/Techreport3002
20. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexity boundaries for Horn description logics. In: Proc. 22nd AAAI Conf. on Artificial Intelligence (AAAI'07), pp. 452–457. AAAI Press, Menlo Park (2007)
21. Krötzsch, M., Rudolph, S., Hitzler, P.: Description logic rules. In: Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N. (eds.) Proc. 18th European Conf. on Artificial Intelligence (ECAI'08), pp. 80–84. IOS Press, Amsterdam (2008)
22. Krötzsch, M., Rudolph, S., Hitzler, P.: ELP: Tractable rules for OWL 2. In: Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 649–664. Springer, Heidelberg (2008)
23. Levy, A.Y., Rousset, M.C.: Combining Horn rules and description logics in CARIN. Artificial Intelligence 104(1-2), 165–209 (1998)
24. Monk, J.: Mathematical Logic. Graduate Texts in Mathematics. Springer, Heidelberg (1976)
25. Motik, B., Horrocks, I., Rosati, R., Sattler, U.: Can OWL and logic programming live together happily ever after? In: Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 501–514. Springer, Heidelberg (2006)
26. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: Veloso, M.M. (ed.) Proc. 20th Int. Joint Conf. on Artificial Intelligence (IJCAI'07), pp. 477–482 (2007)
27. Motik, B., Sattler, U., Studer, R.: Query answering for OWL DL with rules. Journal of Web Semantics 3(1), 41–60 (2005)
28. Rosati, R.: $\mathcal{DL}$+*log*: A tight integration of description logics and disjunctive datalog. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06), pp. 68–78. AAAI Press, Menlo Park (2006)
29. Volz, R.: Web Ontology Reasoning with Logic Databases. Ph.D. thesis, Universität Karlsruhe (TH), Germany (2004)

Inconsistency-Tolerant Semantics for Description Logics

Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo

Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza",
Via Ariosto 25, 00185 Roma, Italy
lastname@dis.uniroma1.it

Abstract. We address the problem of dealing with inconsistencies in Description Logic (DL) knowledge bases. Our general goal is both to study DL semantical frameworks which are inconsistency-tolerant, and to devise techniques for answering unions of conjunctive queries posed to DL knowledge bases under such inconsistency-tolerant semantics. Our work is inspired by the approaches to consistent query answering in databases, which are based on the idea of living with inconsistencies in the database, but trying to obtain only consistent information during query answering, by relying on the notion of database repair. We show that, if we use the notion of repair studied in databases, inconsistency-tolerant query answering is intractable, even for the simplest form of queries. Therefore, we study different variants of the repair-based semantics, with the goal of reaching a good compromise between expressive power of the semantics and computational complexity of inconsistency-tolerant query answering.

1 Introduction

It is well-known that inconsistency causes severe problems in classical logic. In particular, since an inconsistent logical theory has no model, it logically implies every formula, and, therefore, query answering on an inconsistent knowledge base becomes meaningless. In this paper, we address the problem of dealing with inconsistencies in Description Logic (DL) knowledge bases. Our general goal is both to study DL semantical frameworks which are inconsistency-tolerant, and to devise techniques for answering unions of conjunctive queries posed to DL knowledge bases under such inconsistency-tolerant semantics.

A DL knowledge base is constituted by two components, called the TBox and the ABox, respectively [1]. Intuitively, the TBox includes axioms sanctioning general properties of concepts and relations (such as Dog isa Animal), whereas the ABox contains axioms asserting properties of instances of concepts and relations (such as Bob is an instance of Dog). The various DLs differ in the language (set of constructs) used to express such axioms. We are particularly interested in using DLs for the so-called "ontology-based data access" [13] (ODBA), where a DL TBox acts as an ontology used to access a set of data sources. Since it is often the case that, in this setting, the size of the data at the sources largely exceeds the size of the ontology, DLs where query answering is tractable with respect to the size of the ABox have been studied recently. In this paper, we will consider DLs specifically tailored towards ODBA, in particular DLs of the *DL-Lite* family [4], where query answering can be done efficiently with respect to the size of the ABox.

Depending on the expressive power of the underlying language, the TBox alone might be inconsistent, or the TBox might be consistent, but the axioms in the ABox might contradict the axioms in the TBox. Since in ODBA the ontology is usually represented as a consistent TBox, whereas the data at the sources do not necessarily conform to the ontology, the latter situation is the one commonly occurring in practice. Therefore, our study is carried out under the assumption that the TBox is consistent, and inconsistency may arise between the ABox and the TBox (inconsistencies in the TBox are considered, e.g., in [12,9,8,14,11]).

There are many approaches for devising inconsistency-tolerant inference systems [2], originated in different areas, including Logic, Artificial Intelligence, and Databases. Our work is especially inspired by the approaches to consistent query answering in databases [5], which are based on the idea of living with inconsistencies (i.e., data that do not satisfy the integrity constraints) in the database, but trying to obtain only consistent information during query answering. But how can one obtain consistent information from an inconsistent database? The main tool used for this purpose is the notion of database repair: a repair of a database contradicting a set of integrity constraints is a database obtained by applying a minimal set of changes which restore consistency. In general, there are many possible repairs for a database D, and, therefore, the approach sanctions that what is consistently true in D is simply what is true in all possible repairs of D. Thus, inconsistency-tolerant query answering amounts to compute the tuples that are answers to the query in all possible repairs.

In [10], a semantics for inconsistent knowledge bases expressed in *DL-Lite* has been proposed, based on the notion of repair. More specifically, an ABox $\mathcal{A}'$ is a repair of the knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the TBox and $\mathcal{A}$ is the ABox, if it is consistent with $\mathcal{T}$, and there exists no ABox consistent with $\mathcal{T}$ that is "closer" to $\mathcal{A}$, where an ABox $\mathcal{A}''$ is closer to $\mathcal{A}$ than $\mathcal{A}'$ if $\mathcal{A} \cap \mathcal{A}''$ is a proper superset of $\mathcal{A} \cap \mathcal{A}'$. In this paper, we call such semantics the *ABox Repair (AR) semantics*, and we show that for the DLs of the *DL-Lite* family, inconsistency-tolerant query answering under such a semantics is coNP-complete even for ground atomic queries, thus showing that inconsistency-tolerant instance checking is already intractable. For this reason, we propose a variant of the AR-semantics, based on the idea that inconsistency-tolerant query answering should be done by evaluating the query over the intersection of all AR-repairs. The new semantics, called the *Intersection ABox Repair (IAR) semantics*, is an approximation of the AR-semantics, and it enjoys a desirable property, namely that inconsistency-tolerant query answering is polynomially tractable.

Both the AR-semantics and the IAR-semantics suffer from a drawback. Suppose that $\mathcal{K}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ differs from the inconsistent knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, simply because $\mathcal{A}'$ includes assertions that logically follow, using $\mathcal{T}$, from a consistent subset of $\mathcal{A}$. This implies that $\mathcal{K}'$ is also inconsistent, and one would expect that the repairs of $\mathcal{K}'$ and the repairs of $\mathcal{K}$ coincide. On the contrary, since the AR-semantics is not independent from the form of the knowledge base, one can show that, in general, inconsistency-tolerant query answering in the two knowledge bases yields different results. To overcome this drawback, we propose a new variant of the AR-semantics, called the *Closed ABox Repair (CAR) semantics*, that essentially considers only repairs that are "closed" with respect to the knowledge represented by the TBox. We show that, while inconsistency-tolerant instance checking is tractable under this new semantics, query answering is coNP-complete for unions of conjunctive queries. For this reason, we also study the "intersection-based" version of the CAR-semantics, called the *Intersection Closed ABox Repair ($ICAR$) semantics*, showing that it is an approximation of the CAR-semantics, and that inconsistency-tolerant query answering under this new semantics is again polynomially tractable.

The paper is organized as follows. In Section 2 we briefly describe the DL we use in our work. In Section 3 we present the various inconsistency-tolerant semantics we have studied in our investigation. In Section 4 we present the complexity results about such semantics, in terms of both lower bounds and upper bounds. Finally, Section 5 concludes the paper.

2 Preliminaries

Description Logics (DLs) [1] are logics that represent the domain of interest in terms of *concepts*, denoting sets of objects, *value-domains*, denoting sets of values, *attributes*, denoting binary relations between objects and values, and *roles*, denoting binary relations over objects. DL expressions are built starting from an alphabet Γ of symbols for atomic concepts, atomic value-domains, atomic attributes, atomic roles, and object and value constants. We denote by Γ_O the set of object constants, and by Γ_V the set of value constants. Complex expressions are constructed starting from atomic elements, and applying suitable constructs. Different DLs allow for different constructs.

A DL knowledge base (KB) is constituted by two main components: a *TBox* (i.e.,"Terminological Box"), which contains a set of universally quantified assertions stating general properties of concepts and roles, thus representing intensional knowledge of the domain, and an *ABox* (i.e.,"Assertional Box"), which is constituted by assertions on individual objects, thus specifying extensional knowledge. Again, different DLs allow for different kinds of TBox and/or ABox assertions.

Formally, if $\mathcal{L}$ is a DL, then an $\mathcal{L}$-knowledge base $\mathcal{K}$ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox expressed in $\mathcal{L}$ and $\mathcal{A}$ is a ABox. In this paper we assume that the ABox assertions are *atomic*, i.e., they involve only atomic concepts, attributes and roles. The alphabet of $\mathcal{K}$, denoted by $\Gamma_{\mathcal{K}}$, is the set of symbols from Γ occurring in $\mathcal{T}$ and $\mathcal{A}$. The semantics of a DL knowledge base is given in terms of first-order (FOL) interpretations (cf. [1]). We denote with $Mod(\mathcal{K})$ the set of models of $\mathcal{K}$, i.e., the set of FOL interpretations that satisfy all the assertions in $\mathcal{T}$ and $\mathcal{A}$, where the definition of satisfaction depends on the kind of expressions and assertions in the specific DL language in which $\mathcal{K}$ is specified. As usual, a KB $\mathcal{K}$ is said to be *satisfiable* if it admits at least one model, i.e., if $Mod(\mathcal{K}) \neq \emptyset$, and $\mathcal{K}$ is said to *entail* a First-Order Logic (FOL) sentence ϕ, denoted $\mathcal{K} \models \phi$, if $\phi^{\mathcal{I}} = \texttt{true}$ for all $\mathcal{I} \in Mod(\mathcal{K})$.

We now provide some details about the DL *DL-Lite*$_{\mathcal{A}}$, a member of the *DL-Lite* family [4]. This is a family of tractable DLs particularly suited for dealing with

KBs with very large ABoxes, and is at the basis of OWL 2 QL, one of the profiles of OWL 2, the official ontology language of the World Wide Web Consortium (W3C). In *DL-Lite*$_{\mathcal{A}}$, the alphabet Γ is partitioned into 6 subsets, for object constants, value constants, and atomic concept, value-domain, attribute, role symbols, respectively. Concept, role, attribute, and value-domain expressions are formed according to the following syntax:

$$\begin{array}{ll} B \longrightarrow A \mid \exists Q \mid \delta(U) & E \longrightarrow \rho(U) \\ C \longrightarrow B \mid \neg B & F \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n \\ Q \longrightarrow P \mid P^- & V \longrightarrow U \mid \neg U \\ R \longrightarrow Q \mid \neg Q & \end{array}$$

where A, P, and U are symbols in Γ denoting an atomic concept, an atomic role and an atomic attribute respectively, $\top_D$ is the universal value-domain, and $T_1, \ldots, T_n$ are symbols in Γ for atomic value-domains. The expression P^- denotes the inverse of an atomic role, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values, $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects.

A *DL-Lite*$_{\mathcal{A}}$ KB is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the TBox and $\mathcal{A}$ the ABox. The TBox $\mathcal{T}$ is a finite set of assertions of the form

$$B \sqsubseteq C \qquad Q \sqsubseteq R \qquad E \sqsubseteq F \qquad U \sqsubseteq V \qquad (\mathsf{funct}\ Q) \qquad (\mathsf{funct}\ U)$$

From left to right, the first four assertions denote inclusions between concepts, roles, value-domains, and attributes, respectively. The last two assertions denote functionality on roles and on attributes. *DL-Lite*$_{\mathcal{A}}$ TBoxes are subject to the restriction that roles and attributes occurring in functionality assertions cannot be specialized (i.e., they cannot occur in the right-hand side of inclusions).

A *DL-Lite*$_{\mathcal{A}}$ ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$, $P(a, b)$, and $U(a, v)$, where A, P, and U are as above, a and b are object constants from Γ_O, and v is a value constant from Γ_V.

Example 1. We consider a simple *DL-Lite*$_{\mathcal{A}}$ knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ describing the "Formula One Teams" domain, where the TBox $\mathcal{T}$ is constituted by the following assertions:

$$\begin{array}{lll} \mathsf{Mechanic} \sqsubseteq \mathsf{TeamMember} & \mathsf{Driver} \sqsubseteq \mathsf{TeamMember} & \mathsf{Driver} \sqsubseteq \neg\mathsf{Mechanic} \\ \exists\mathsf{drives} \sqsubseteq \mathsf{Driver} & \exists\mathsf{drives}^- \sqsubseteq \mathsf{Car} & (\mathsf{funct}\ \mathsf{drives}) \end{array}$$

In words, $\mathcal{T}$ specifies that drivers and mechanics are team members, but drivers are not mechanics (note that $\mathsf{Driver} \sqsubseteq \neg\mathsf{Mechanic}$ is called a disjointness assertion, because it states that the two concepts Driver and $\mathsf{Mechanic}$ are dosjoint). Moreover, the role drives has Driver as domain and Car as range, and it is also functional, i.e., every driver can drive at most one car. The ABox $\mathcal{A} = \{\mathsf{Driver}(felipe), \mathsf{TeamMember}(felipe), \mathsf{drives}(felipe, ferrari)\}$ asserts that $felipe$ is both a driver and a team member, and that he drives the car $ferrari$.

The semantics of a *DL-Lite*$_{\mathcal{A}}$ KB is given in terms of FOL interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is the interpretation domain and $\cdot^{\mathcal{I}}$ is the interpretation function. In particular,

$\Delta^{\mathcal{I}}$ is a non-empty set partitioned into Δ_V and $\Delta_O^{\mathcal{I}}$, where $\Delta_O^{\mathcal{I}}$ is the subset of Δ used to interpret object constants in Γ_O, and Δ_V is the subset of Δ used to interpret data values. In other words, for every $c \in \Gamma_O$, $c^{\mathcal{I}} \in \Delta_O$ and for every $f \in \Gamma_V$, $f^{\mathcal{I}} \in \Delta_V$. The interpretation function $\cdot^{\mathcal{I}}$ is defined as follows.

- For every $c \in \Gamma_O$, $c^{\mathcal{I}} \in \Delta_O$, and for every $f \in \Gamma_V$, $f^{\mathcal{I}} \in \Delta_V$.
- For all $d_1, d_2 \in \Gamma_O \cup \Gamma_V$, $d_1^{\mathcal{I}} \neq d_2^{\mathcal{I}}$ (i.e., interpretations for *DL-Lite* follow the unique name assumption).
- For all $1 \leq i \leq n$, T_i is an unbound set of values.
- The following equations are satisfied by $\cdot^{\mathcal{I}}$:

$$\begin{array}{llll}
A^{\mathcal{I}} & \subseteq \Delta_O^{\mathcal{I}} & P^{\mathcal{I}} & \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \\
(\delta(U))^{\mathcal{I}} & = \{ o \mid \exists v.\, (o,v) \in U^{\mathcal{I}} \} & (P^-)^{\mathcal{I}} & = \{ (o,o') \mid (o',o) \in P^{\mathcal{I}} \} \\
(\exists Q)^{\mathcal{I}} & = \{ o \mid \exists o'.\, (o,o') \in Q^{\mathcal{I}} \} & (\neg Q)^{\mathcal{I}} & = (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}} \\
(\neg B)^{\mathcal{I}} & = \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}} & U^{\mathcal{I}} & \subseteq \Delta_O^{\mathcal{I}} \times \Delta_V \\
(\rho(U))^{\mathcal{I}} & = \{ v \mid \exists o.\, (o,v) \in U^{\mathcal{I}} \} & (\neg U)^{\mathcal{I}} & = (\Delta_O^{\mathcal{I}} \times \Delta_V) \setminus U^{\mathcal{I}} \\
T_i^{\mathcal{I}} & \subseteq \top_D^{\mathcal{I}} = \Delta_V \ (1 \leq i \leq n) & T_i \cap T_j & = \emptyset \quad (1 \leq i,j \leq n)
\end{array}$$

An interpretation $\mathcal{I}$ satisfies a concept (resp., role) inclusion assertion $B \sqsubseteq C$ (resp., $Q \sqsubseteq R$) if $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ (resp., $Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$), and satisfies a role functionality assertion $(\mathsf{funct}\ Q)$ if, for each $o, o', o'' \in \Delta_O^{\mathcal{I}}$, we have that $(o,o') \in Q^{\mathcal{I}}$ and $(o,o'') \in Q^{\mathcal{I}}$ implies $o' = o''$. The semantics for attribute and value-domain inclusion assertions, and for functionality assertions over attributes can be defined analogously. Finally, $\mathcal{I}$ satisfies ABox assertions $A(a)$, $P(a,b)$ and $U(a,v)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$ and $(a^{\mathcal{I}}, v^{\mathcal{I}}) \in U^{\mathcal{I}}$ respectively.

In the following, we are interested in particular in the problem of answering queries posed to a *DL-Lite*$_{\mathcal{A}}$-KB. More specifically, we deal with the problem of establishing whether a *DL-Lite*$_{\mathcal{A}}$ KB entails a *boolean union of conjunctive queries (UCQ)*, i.e., a first order sentence of the form $\exists \boldsymbol{y_1}.conj_1(\boldsymbol{y_1}) \vee \cdots \vee \exists \boldsymbol{y_n}.conj_n(\boldsymbol{y_n})$, where $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$ are terms (i.e., constants or variables), and each $conj_i(\boldsymbol{y_i})$ is a conjunction of atoms of the form $A(z)$, $P(z,z')$ and $U(z,z')$ where A is an atomic concept, P is an atomic role and U is an attribute name, and z, z' are terms. Notice that all the results we achieve about this reasoning task can be easily extended in the standard way to the presence of free variables in queries (see e.g. [7]).

In the rest of this paper we will focus on the *data complexity* of query answering, i.e., we will measure the computational complexity only with respect to the size of the ABox (which is usually much larger than the TBox and the queries). It follows from the results in [4,13] that query answering in *DL-Lite*$_{\mathcal{A}}$ is in AC_o, which is a complexity class contained in PTIME, and therefore is tractable in data complexity.

3 Inconsistency-Tolerant Semantics

In this section we present our inconsistency-tolerant semantics for DL knowledge bases. As we said in the introduction, we assume that for a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, $\mathcal{T}$ is satisfiable, whereas $\mathcal{A}$ may be inconsistent with $\mathcal{T}$, i.e., the set of models of $\mathcal{K}$ may be empty. The challenge is to provide semantic characterizations for $\mathcal{K}$, which are *inconsistency-tolerant*, i.e., they allow $\mathcal{K}$ to be interpreted with a non-empty set of models even in the case where it is unsatisfiable under the classical first-order semantics.

The inconsistency-tolerant semantics we give below are based on the notion of *repair*. Intuitively, given a DL KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, a repair $\mathcal{A}_R$ for $\mathcal{K}$ is an ABox such that the KB $\langle \mathcal{T}, \mathcal{A}_R \rangle$ is satisfiable under the first-order semantics, and $\mathcal{A}_R$ "minimally" differs from $\mathcal{A}$. Notice that in general not a single, but several repairs may exist, depending on the particular minimality criteria adopted. We consider here different notions of "minimality", which give rise to different inconsistency-tolerant semantics. In all cases, such semantics coincide with the classical first-order semantics when inconsistency does not come into play, i.e., when the KB is satisfiable under standard first-order semantics.

The first notion of repair that we consider can be phrased as follows: a repair $\mathcal{A}_R$ of a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a maximal subset of $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A}_R \rangle$ is satisfiable under the first-order semantics, i.e., there does not exist another subset of $\mathcal{A}$ that strictly contains $\mathcal{A}_R$ and that is consistent with $\mathcal{T}$. Intuitively, each such repair is obtained by throwing away from $\mathcal{A}$ a minimal set of assertions to make it consistent with $\mathcal{T}$. In other words, adding to $\mathcal{A}_R$ another assertion of $\mathcal{A}$ would make the repair inconsistent with $\mathcal{T}$. The formal definition is given below.

Definition 1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL KB. An* ABox Repair (AR) *of $\mathcal{K}$ is a set $\mathcal{A}'$ of membership assertions such that:*

1. *$\mathcal{A}' \subseteq \mathcal{A}$,*
2. *$Mod(\langle \mathcal{T}, \mathcal{A}' \rangle) \neq \emptyset$,*
3. *there exist no $\mathcal{A}''$ such that $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $Mod(\langle \mathcal{T}, \mathcal{A}'' \rangle) \neq \emptyset$.*

The set of AR-repairs for $\mathcal{K}$ is denoted by AR-Rep($\mathcal{K}$).

Based on the above notion of repair, we can now give the definition of ABox repair model.

Definition 2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL KB. An interpretation $\mathcal{I}$ is an* ABox repair model*, or simply an AR*-model*, of $\mathcal{K}$ if there exists $\mathcal{A}' \in$ AR-Rep($\mathcal{K}$) such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{A}' \rangle$. The set of ABox repair models of $\mathcal{K}$ is denoted by AR-Mod($\mathcal{K}$).*

The following notion of consistent entailment is the natural generalization of classical entailment to the ABox repair semantics.

Definition 3. *Let $\mathcal{K}$ be a DL KB, and let ϕ be a first-order sentence. We say that ϕ is AR*-consistently entailed*, or simply AR*-entailed*, by $\mathcal{K}$, written $\mathcal{K} \models_{AR} \phi$, if $\mathcal{I} \models \phi$ for every $\mathcal{I} \in$ AR-Mod($\mathcal{K}$).*

Example 2. Consider the *DL-Lite$_{\mathcal{A}}$* knowledge base $\mathcal{K}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, where $\mathcal{T}$ is the TBox of the KB presented in the Example 1, and $\mathcal{A}'$ is the ABox constituted by the set of assertions:

$$\mathcal{A}' = \{\mathsf{Driver}(felipe), \mathsf{Mechanic}(felipe), \mathsf{TeamMember}(felipe), \\ \mathsf{drives}(felipe, ferrari)\}.$$

This ABox states that $felipe$ is a team member and that he is both a driver and a mechanic. Notice that this implies that $felipe$ drives $ferrari$ and that $ferrari$ is a car. It is easy to see that $\mathcal{K}$ is unsatisfiable, since $felipe$ violates the disjointness between driver and mechanic.

The set *AR-Rep*$(\mathcal{K}')$ is constituted by the set of $\mathcal{T}$*-consistent* ABoxes:

$$AR\text{-}rep_1 = \{\mathsf{Driver}(felipe), \mathsf{drives}(felipe, ferrari), \mathsf{TeamMember}(felipe)\};$$
$$AR\text{-}rep_2 = \{\mathsf{Mechanic}(felipe), \mathsf{TeamMember}(felipe)\}.$$

Note that to obtain $AR\text{-}rep_1$ it is sufficient to remove $\mathsf{Mechanic}(felipe)$ from $\mathcal{A}$, whereas to obtain $AR\text{-}rep_2$, we need to remove from $\mathcal{A}$ both $\mathsf{Driver}(felipe)$, which is obvious, and $\mathsf{Driver}(felipe, ferrari)$, which, together with the TBox assertion $\exists\mathsf{drives} \sqsubseteq \mathsf{Driver}$, implies $\mathsf{Driver}(felipe)$.

The AR-semantics given above in fact coincides with the inconsistency-tolerant semantics for DL KBs presented in [10], and with the loosely-sound semantics studied in [3] in the context of inconsistent databases. Although this semantics can be considered to some extent the natural choice for the setting we are considering, since each ABox repair stays as close as possible to the original ABox, it has the characteristic to be dependent from the form of the knowledge base. Suppose that $\mathcal{K}'' = \langle\mathcal{T}, \mathcal{A}''\rangle$ differs from the inconsistent knowledge base $\mathcal{K}' = \langle\mathcal{T}, \mathcal{A}'\rangle$, simply because $\mathcal{A}''$ includes assertions that logically follow, using $\mathcal{T}$, from a consistent subset of $\mathcal{A}$ (implying that $\mathcal{K}''$ is also inconsistent). One could argue that the repairs of $\mathcal{K}''$ and the repairs of $\mathcal{K}'$ should coincide. Conversely, the next example shows that, in the AR-semantics the two sets of repairs are generally different.

Example 3. Consider the KB $\mathcal{K}'' = \langle\mathcal{T}, \mathcal{A}''\rangle$, where $\mathcal{T}$ is the same as in $\mathcal{K}' = \langle\mathcal{T}, \mathcal{A}'\rangle$ of Example 2, and the ABox $\mathcal{A}''$ is as follows:

$$\mathcal{A}'' = \{\mathsf{Driver}(felipe), \mathsf{Mechanic}(felipe), \mathsf{TeamMember}(felipe), \mathsf{Car}(ferrari), \mathsf{drives}(felipe, ferrari)\}.$$

Notice that $\mathcal{A}''$ can be obtained by adding $\mathsf{Car}(ferrari)$ to $\mathcal{A}'$. Since $\mathsf{Car}(ferrari)$ is entailed by the KB $\langle\mathcal{T}, \{\mathsf{drives}(felipe, ferrari)\}\rangle$, i.e., a KB constituted by the TBox $\mathcal{T}$ of $\mathcal{K}'$ and a subset of $\mathcal{A}'$ that is consistent with $\mathcal{T}$, one intuitively would expect that $\mathcal{K}'$ and $\mathcal{K}''$ have the same repairs under the AR-semantics. This is however not the case, since we have that *AR-Rep*$(\mathcal{K}'')$ is formed by:

$$AR\text{-}rep_3 = \{\mathsf{Driver}(felipe), \mathsf{drives}(felipe, ferrari), \mathsf{TeamMember}(felipe), \mathsf{Car}(ferrari)\};$$
$$AR\text{-}rep_4 = \{\mathsf{Mechanic}(felipe), \mathsf{TeamMember}(felipe), \mathsf{Car}(ferrari)\}.$$

Let us finally consider the ground sentence $\mathsf{Car}(ferrari)$. It is easy to see that $\mathsf{Car}(ferrari)$ is AR-entailed by the KB $\mathcal{K}''$ but it is not AR-entailed by the KB $\mathcal{K}'$.

Depending on the particular scenario, and the specific application at hand, the above behavior might be considered incorrect. This motivates the definition of a new semantics that does not present such a characteristic. According to this new semantics, that we call *Closed ABox Repair*, the repairs take into account not only the assertions explicitly included in the ABox, but also those that are implied, through the TBox, by at least one subset of the ABox that is consistent with the TBox.

To formalize the above idea, we need some preliminary definitions. Given a DL KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we denote with $HB(\mathcal{K})$ the *Herbrand Base of* $\mathcal{K}$, i.e. the set of ABox assertions that can be built over the alphabet of $\Gamma_{\mathcal{K}}$. Then we define the *consistent logical consequences of* $\mathcal{K}$ as the set $clc(\mathcal{K}) = \{\alpha \mid \alpha \in HB(\mathcal{K})$ and there exists $S \subseteq \mathcal{A}$ such that $Mod(\langle \mathcal{T}, S \rangle) \neq \emptyset$ and $\langle \mathcal{T}, S \rangle \models \alpha\}$. With the above notions in place we can now give the definition of Closed ABox Repair.

Definition 4. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL KB. A* Closed ABox Repair (CAR) *for $\mathcal{K}$ is a set $\mathcal{A}'$ of membership assertions such that:*

1. *$\mathcal{A}' \subseteq clc(\mathcal{K})$,*
2. *$Mod(\langle \mathcal{T}, \mathcal{A}' \rangle) \neq \emptyset$,*
3. *there exist no $\mathcal{A}'' \subseteq clc(\mathcal{K})$ such that*
   (a) $Mod(\langle \mathcal{T}, \mathcal{A}'' \rangle) \neq \emptyset$, and
   (b) it is either $\mathcal{A}'' \cap \mathcal{A} \supset \mathcal{A}' \cap \mathcal{A}$ or $\mathcal{A}'' \cap \mathcal{A} = \mathcal{A}' \cap \mathcal{A}$ and $\mathcal{A}'' \supset \mathcal{A}'$.

The set of CAR-repairs for $\mathcal{K}$ is denoted by $CAR\text{-}Rep(\mathcal{T}, \mathcal{A})$.

Intuitively, a CAR-repair is a subset of $clc(\mathcal{K})$ consistent with $\mathcal{T}$ that "maximally preserves" the ABox $\mathcal{A}$. In particular, condition 3 states that we prefer $\mathcal{A}'$ to any other $\mathcal{A}_R \subseteq clc(\mathcal{K})$ consistent with $\mathcal{T}$ such that $\mathcal{A}_R \cap \mathcal{A} \subset \mathcal{A}' \cap \mathcal{A}$ (i.e., $\mathcal{A}_R$ maintains a smaller subset of $\mathcal{A}$ with respect to $\mathcal{A}'$). Then, among those $\mathcal{A}_R$ having the same intersection with $\mathcal{A}$, we prefer the ones that contain as much assertions of $clc(\mathcal{K})$ as possible.

The set of CAR-models of a KB $\mathcal{K}$, denoted $CAR\text{-}Mod(\mathcal{K})$, is defined analogously to AR-models (cf. Definition 2). Also, CAR-entailment, denoted $\models_{CAR}$, is analogous to AR-entailment (cf. Definition 3).

Example 4. Consider the two KBs $\mathcal{K}'$ and $\mathcal{K}''$ presented in the Example 2 and Example 3. It is easy to see that both $CAR\text{-}Rep(\mathcal{K}')$ and $CAR\text{-}Rep(\mathcal{K}'')$ are constituted by the two sets below:

$$\begin{aligned} CAR\text{-}rep_1 = \{&\mathsf{Driver}(felipe), \mathsf{drives}(felipe, ferrari), \mathsf{TeamMember}(felipe),\\ &\mathsf{Car}(ferrari)\};\\ CAR\text{-}rep_2 = \{&\mathsf{Mechanic}(felipe), \mathsf{TeamMember}(felipe), \mathsf{Car}(ferrari)\}. \end{aligned}$$

It follows that both $\mathcal{K}'$ and $\mathcal{K}''$ CAR-*entail* the ground sentence $\mathsf{Car}(ferrari)$, differently from what happen under the AR-semantics, as showed in Example 3.

The above example shows also that there are sentences entailed by a KB under the CAR-semantics that are not entailed under the AR-semantics. Conversely, we can show that the AR-semantics is a sound approximation of the CAR-semantics, i.e., for any KB $\mathcal{K}$ $CAR\text{-}Mod(\mathcal{K}) \subseteq AR\text{-}Mod(\mathcal{K})$, implying that the logical consequences of $\mathcal{K}$ under the AR-semantics are contained in the logical consequences of $\mathcal{K}$ under the CAR-semantics, as stated by the following theorem.

Theorem 1. *Let $\mathcal{K}$ be a DL KB, and ϕ a first-order sentence. Then, $\mathcal{K} \models_{AR} \phi$ implies $\mathcal{K} \models_{CAR} \phi$.*

$$\begin{tikzcd}
 & \textbf{\textit{CAR}} & \\
\textbf{\textit{AR}} \arrow[ur] & & \textbf{\textit{ICAR}} \arrow[ul] \\
 & \textbf{\textit{IAR}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Fig. 1. Partial order over the inconsistency tolerant semantics

As we will see in the next section, entailment of a union of conjunctive queries from a KB $\mathcal{K}$ is intractable both under the AR-semantics and the CAR-semantics. Since this can be an obstacle in the practical use of such semantics, we introduce here approximations of the two semantics, under which we will show in the next section that entailment of unions of conjunctive queries is polynomial. In both cases, the approximation consists in taking as unique repair the intersection of the AR-repairs and of the CAR-repairs, respectively. This actually corresponds to follow the WIDTIO (When you are in doubt throw it out) approach, proposed in the area of belief revision and update [15,6].

Definition 5. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL KB. An* Intersection ABox Repair (IAR) *for $\mathcal{K}$ is the set $\mathcal{A}'$ of membership assertions such that $\mathcal{A}' = \bigcap_{\mathcal{A}_i \in \textit{AR-Rep}(\mathcal{K})} \mathcal{A}_i\}$. The (singleton) set of IAR-repairs for $\mathcal{K}$ is denoted by IAR-Rep$(\mathcal{K})$.*

Analogously, we give below the definition of Intersection Closed ABox Repair.

Definition 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL KB. An* Intersection Closed ABox Repair ($ICAR$) *for $\mathcal{K}$ is the set $\mathcal{A}'$ of membership assertions such that $\mathcal{A}' = \bigcap_{\mathcal{A}_i \in \textit{CAR-Rep}(\mathcal{K})} \mathcal{A}_i\}$. The (singleton) set of $ICAR$-repairs for $\mathcal{K}$ is denoted by ICAR-Rep$(\mathcal{K})$.*

The sets *IAR-Mod*$(\mathcal{K})$ and *ICAR-Mod*$(\mathcal{K})$ of IAR-models and $ICAR$-models, respectively, and the notions of IAR-entailment and $ICAR$-entailment are defined as usual (cf. Definition 2 and Definition 3).

Example 5. Consider the KB $\mathcal{K}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ presented in Example 2. Then *IAR-Rep*$(\mathcal{K}')$ is the singleton formed by the ABox $IAR\text{-}rep = AR\text{-}rep_1 \cap AR\text{-}rep_2 = \{\mathsf{TeamMember}(felipe)\}$. In turn, referring to Example 4, *ICAR-Rep*$(\mathcal{K}')$ is the singleton formed by the ABox $ICAR\text{-}rep_1 = CAR\text{-}rep_1 \cap CAR\text{-}rep_2$ = $\{$TeamMember(felipe),Car(ferrari)$\}$.

It is not difficult to show that the IAR-semantics is a sound approximation of the AR-semantics, and that the $ICAR$-semantics is a sound approximation of the CAR-semantics. It is also easy to see that the converse is not true in general. For instance, the sentence $\mathsf{Driver}(felipe)$ is entailed by $\mathcal{K} = \langle \mathcal{T}, \{\mathsf{drives}(felipe, ferrari), \mathsf{drives}(felipe, mcLaren)\}\rangle$, where $\mathcal{T}$ is the TBox of Example 1, under the AR-semantics, but it is not entailed under the IAR-semantics.

Furthermore, an analogous of Theorem 1 holds also for the "intersection" semantics.

Theorem 2. *Let $\mathcal{K}$ be a DL KB, and ϕ a first-order sentence. Then, $\mathcal{K} \models_{IAR} \alpha$ implies $\mathcal{K} \models_{ICAR} \alpha$.*

Also in this case one can easily see that the converse implication does not hold. It is sufficient to look again at Example 5, where $\mathsf{Car}(ferrari)$ is entailed by $\mathcal{K}'$ under the $ICAR$-semantics, but it is not entailed under the IAR-semantics.

From all the above results it follows that the AR-, CAR-, IAR-, and $ICAR$-semantics form a partial order, where the CAR-semantics is the upper bound, the IAR-semantics is the lower bound, whereas the $ICAR$-semantics and the AR-semantics are incomparable (see Figure 1). In other words, the IAR-semantics is a sound approximation of all the semantics, while the CAR-semantics is the one which is able to derive the largest set of conclusions from a KB. It can also easily be shown that the AR-semantics and the $ICAR$-semantics are incomparable.

4 Reasoning

In this section we study reasoning in the inconsistency-tolerant semantics introduced in the previous section. In particular, we analyze the problem of UCQ entailment under such semantics. We will also consider instance checking, which is a restricted form of UCQ entailment. As we said before, in our analysis we will focus on data complexity.

We start by considering the AR-semantics. It is known that UCQ entailment is intractable under this semantics [10]. Here, we strengthen this result, and show that instance checking under the AR-semantics is already coNP-hard in data complexity even if the KB is expressed in *DL-Lite*$_{core}$. We recall that *DL-Lite*$_{core}$ is the least expressive logic in the *DL-Lite* family, as it only allows for concept expressions of the form $C ::= A|\exists R|\exists R^-$, and for TBox assertions of the form $C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \neg C_2$ (for more details, see [4]).

Theorem 3. *Let $\mathcal{K}$ be a DL-Lite$_{core}$ KB and let α be an ABox assertion. Deciding whether $\mathcal{K} \models_{AR} \alpha$ is coNP-complete with respect to data complexity.*

Proof. Membership in coNP follows from coNP-completeness of UCQ entailment under AR-semantics [10, Theorem 1].

We prove hardness with respect to coNP by reducing satisfiability of a 3-CNF formula to the complement of instance checking.

Let ϕ be a 3-CNF, i.e., a formula of the form $\phi = c_1 \wedge \ldots \wedge c_k$ where $c_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$ for every i such that $1 \leq i \leq k$ (every ℓ_j^i is a propositional literal). Let $a_1, \ldots, a_n$ be the propositional variable symbols occurring in ϕ.

We define the following TBox $\mathcal{T}$ (which does not depend on ϕ):

$$\begin{array}{lll}
\exists R^- \sqsubseteq \neg \exists LT_1^- & \exists R \sqsubseteq \mathit{Unsat} & \exists LT_2 \sqsubseteq \neg \exists LF_2 \\
\exists R^- \sqsubseteq \neg \exists LF_1^- & \exists LT_1 \sqsubseteq \neg \exists LF_1 & \exists LT_2 \sqsubseteq \neg \exists LF_3 \\
\exists R^- \sqsubseteq \neg \exists LT_2^- & \exists LT_1 \sqsubseteq \neg \exists LF_2 & \exists LF_2 \sqsubseteq \neg \exists LT_3 \\
\exists R^- \sqsubseteq \neg \exists LF_2^- & \exists LT_1 \sqsubseteq \neg \exists LF_3 & \exists LT_3 \sqsubseteq \neg \exists LF_3 \\
\exists R^- \sqsubseteq \neg \exists LT_3^- & \exists LF_1 \sqsubseteq \neg \exists LT_2 & \\
\exists R^- \sqsubseteq \neg \exists LF_3^- & \exists LF_1 \sqsubseteq \neg \exists LT_3 &
\end{array}$$

Then, we define the following ABox $\mathcal{A}_\phi$ (which depends on ϕ):

$$\{R(a,c_1),\dots,R(a,c_k)\} \cup \bigcup_{i=1}^{k} \bigcup_{j=1}^{3} \{\mathit{Polarity}(\ell_j^i)(\mathit{Atom}(\ell_j^i), c_i)\}$$

where: (i) $\mathit{Polarity}(\ell_j^i) = \mathit{LT}_j$ if ℓ_j^i is a positive literal, while $\mathit{Polarity}(\ell_j^i) = \mathit{LF}_j$ if ℓ_j^i is a negative literal; (ii) $\mathit{Atom}(\ell_j^i)$ denotes the propositional variable symbol occurring in literal ℓ_j^i. For instance, if $\phi = (a_1 \vee \neg a_2 \vee a_3) \wedge (\neg a_3 \vee a_4 \vee \neg a_1)$, the ABox $\mathcal{A}_\phi$ is the following:

$$\begin{array}{l}\{R(a,c_1), R(a,c_2)\} \cup \\ \{LT_1(a_1,c_1), LF_2(a_2,c_1), LT_3(a_3,c_1), LF_1(a_3,c_2), LT_2(a_4,c_2), LF_3(a_1,c_2)\}\end{array}$$

We now prove that $\langle \mathcal{T}, \mathcal{A}_\phi \rangle \not\models_{AR} \mathit{Unsat}(a)$ iff ϕ is satisfiable.

First, suppose ϕ is satisfiable, and let $\mathcal{I}$ be a model for ϕ. Then, let $\mathcal{A}'$ be the following subset of $\mathcal{A}_\phi$:

$$\begin{array}{l}\{LT_j(a_i,c_h) \mid LT_j(a_i,c_h) \in \mathcal{A}_\phi \text{ and } \mathcal{I} \models a_i\} \cup \\ \{LF_j(a_i,c_h) \mid LF_j(a_i,c_h) \in \mathcal{A}_\phi \text{ and } \mathcal{I} \not\models a_i\}\end{array}$$

It is immediate to verify that $\mathcal{A}'$ is a maximal $\mathcal{T}$-consistent subset of $\mathcal{A}_\phi$: in particular, since $\mathcal{I} \models \phi$, if we add any $R(a,c_i)$ to $\mathcal{A}'$, the resulting ABox is inconsistent. Therefore, $\mathcal{A}'$ is an AR-repair of $\langle \mathcal{T}, \mathcal{A}_\phi \rangle$. Moreover, since no $R(a,c_i)$ belongs to $\mathcal{A}'$, it follows that $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \mathit{Unsat}(a)$, which implies that $\langle \mathcal{T}, \mathcal{A}_\phi \rangle \not\models_{AR} \mathit{Unsat}(a)$.

Conversely, suppose $\langle \mathcal{T}, \mathcal{A}_\phi \rangle \not\models_{AR} \mathit{Unsat}(a)$. Then, there exists an AR-repair $\mathcal{A}'$ of $\langle \mathcal{T}, \mathcal{A}_\phi \rangle$ such that $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \mathit{Unsat}(a)$. Of course, no $R(a,c_i)$ belongs to $\mathcal{A}'$. Now, let $\mathcal{I}$ be the propositional interpretation defined as follows: for every a_i, if there exists a fact of the form $LT_j(a_i,c_h) \in \mathcal{A}'$, then $\mathcal{I} \models a_i$, otherwise $\mathcal{I} \not\models a_i$. It is easy to see that, since no $R(a,c_h)$ belongs to $\mathcal{A}'$, for every c_h either there exists a fact of the form $LT_j(a_i,c_h)$ in $\mathcal{A}'$ or there exists a fact of the form $LF_j(a_i,c_h)$ in $\mathcal{A}'$. In both cases, we get $\mathcal{I} \models c_h$. Consequently, $\mathcal{I} \models \phi$. □

Theorem 3 corrects a wrong result presented in [10, Theorem 6], which asserts tractability of AR-entailment of ABox assertions from KBs specified in *DL-Lite*$_\mathcal{F}$, a superset of *DL-Lite*$_{core}$. It turns out that, while the algorithm presented in [10] (on which the above cited Theorem 6 was based) is actually unable to deal with general TBoxes, such a technique can be adapted to prove that AR-entailment of ABox assertions is tractable for *DL-Lite*$_\mathcal{A}$ KBs without TBox disjointness assertions.

Next, we focus on the CAR-semantics, and show that UCQ entailment under this semantics is coNP-hard even if the TBox language is restricted to *DL-Lite*$_{core}$.

Theorem 4. *Let $\mathcal{K}$ be a DL-Lite$_{core}$ KB and let Q be a UCQ. Deciding whether $\mathcal{K} \models_{CAR} Q$ is coNP-complete with respect to data complexity.*

Proof. The proof of coNP-hardness is obtained by a slight modification of the reduction from 3-CNF satisfiability in the proof of Theorem 3. To prove membership in coNP, we first prove that, when $\mathcal{K}$ is a *DL-Lite*$_\mathcal{A}$ KB, *clc*$(\mathcal{K})$ can be computed in polynomial time, which is an immediate consequence of the fact that

$$clc(\mathcal{K}) = \{\alpha \mid \alpha \in HB(\mathcal{K}) \text{ and } \alpha_i \in \mathcal{A} \text{ and } \langle \mathcal{T}, \{\alpha_i\}\rangle \models \alpha \text{ and } \langle \mathcal{T}, \{\alpha_i\}\rangle \text{ satisfiable}\}$$

Now, membership in coNP follows from the following facts: (i) $\langle \mathcal{T}, \mathcal{A}\rangle \models_{CAR} q$ iff $\langle \mathcal{T}, clc(\mathcal{T}, \mathcal{A})\rangle \models_{AR} q$; (ii) $clc(\mathcal{T}, \mathcal{A})$ can be computed in polynomial time; (iii) Theorem 3. □

Notice that, differently from the AR-semantics, the above intractability result for the CAR-semantics does not hold already for the instance checking problem: we will show later in this section that instance checking is indeed tractable under the CAR-semantics.

We now turn our attention to the IAR-semantics, and define the following algorithm *Compute-IAR-Repair* for computing the IAR-repair of a *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{K}$.

```
Algorithm Compute-IAR-Repair($\mathcal{K}$)
input: DL-Lite$_{\mathcal{A}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$
output: DL-Lite$_{\mathcal{A}}$ ABox $\mathcal{A}'$
begin
  let $\mathcal{D} = \emptyset$;
  for each fact $\alpha \in \mathcal{A}$ do
    if $\langle \mathcal{T}, \{\alpha\}\rangle$ unsatisfiable
    then let $\mathcal{D} = \mathcal{D} \cup \{\alpha\}$;
  for each pair of facts $\alpha_1, \alpha_2 \in \mathcal{A} - \mathcal{D}$ do
    if $\langle \mathcal{T}, \{\alpha_1, \alpha_2\}\rangle$ unsatisfiable
    then let $\mathcal{D} = \mathcal{D} \cup \{\alpha_1, \alpha_2\}$;
  return $\mathcal{A} - \mathcal{D}$
end
```

The algorithm is very simple: it computes a set $\mathcal{D}$ of ABox assertions in $\mathcal{A}$ which must be eliminated from the IAR-repair of $\mathcal{K}$.

Lemma 1. *Let $\mathcal{K}$ be a DL-Lite$_{\mathcal{A}}$ KB. Then, IAR-Rep$(\mathcal{K})$ = $\{$Compute-IAR-Repair$(\mathcal{K})\}$.*

Proof. (sketch) The proof is based on the following property, not difficult to verify, which is due to the form of the TBox assertions allowed in *DL-Lite*$_{\mathcal{A}}$: every ABox assertion α that does not belong to at least one AR-repair of $\mathcal{K}$ satisfies one of the following conditions: (i) α is such that the KB $\langle \mathcal{T}, \{\alpha\}\rangle$ is unsatisfiable; (ii) α is such that there exists another ABox assertion α' such that the KB $\langle \mathcal{T}, \{\alpha, \alpha'\}\rangle$ is unsatisfiable and α' does not satisfy the previous condition (i). Therefore, at the end of the execution of the algorithm, the set $\mathcal{D}$ contains every ABox assertion α that does not belong any AR-repair of $\mathcal{K}$. Hence $\mathcal{A} - \mathcal{D}$ is the IAR-repair of $\mathcal{K}$.

The following property, based on the correctness of the previous algorithm, establishes tractability of UCQ entailment under IAR-semantics.

Theorem 5. *Let $\mathcal{K}$ be a DL-Lite$_{\mathcal{A}}$ KB, and let Q be a UCQ. Deciding whether $\mathcal{K} \models_{IAR} Q$ is in PTIME with respect to data complexity.*

Proof. By Lemma 1, the ABox returned by *Compute-IAR-Repair*$(\mathcal{K})$ is the IAR-repair of $\mathcal{K}$. Then, by definition of IAR-semantics, we have that, for every UCQ Q, $\mathcal{K} \models_{IAR} Q$ iff $\langle \mathcal{T}, \mathcal{A}' \rangle \models Q$ where $\mathcal{A}'$ is the IAR-repair of $\mathcal{K}$. From the fact that the algorithm *Compute-IAR-Repair*$(\mathcal{K})$ runs in polynomial time and from tractability of UCQ entailment in *DL-Lite*$_{\mathcal{A}}$ [13], the claim follows. □

We now turn our attention to the $ICAR$-semantics and present the algorithm *Compute-ICAR-Repair* for computing the $ICAR$-repair of a *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{K}$.

```
**Algorithm** Compute-ICAR-Repair($\mathcal{K}$)
**input**: DL-Lite$_{\mathcal{A}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$
**output**: DL-Lite$_{\mathcal{A}}$ ABox $\mathcal{A}'$
**begin**
  Compute clc($\mathcal{K}$);
  **let** $\mathcal{D} = \emptyset$;
  **for each** pair of facts $\alpha_1, \alpha_2 \in$ clc($\mathcal{K}$) **do**
    **if** $\langle \mathcal{T}, \{\alpha_1, \alpha_2\} \rangle$ unsatisfiable
    **then let** $\mathcal{D} = \mathcal{D} \cup \{\alpha_1, \alpha_2\}$;
  **return** clc($\mathcal{K}$) $- \mathcal{D}$
**end**
```

The algorithm is analogous to the previous algorithm *Compute-IAR-Repair*. The main differences are the following: (i) the algorithm *Compute-ICAR-Repair* returns (and operates on) a subset of *clc*$(\mathcal{K})$, while the algorithm *Compute-IAR-Repair* returns a subset of the original ABox $\mathcal{A}$; (ii) differently from the algorithm *Compute-IAR-Repair*, the algorithm *Compute-ICAR-Repair* does not need to eliminate ABox assertions α such that $\langle \mathcal{T}, \{\alpha\} \rangle$ is unsatisfiable, since such facts cannot occur in *clc*$(\mathcal{K})$.

Again, through the algorithm *Compute-ICAR-Repair* it is possible to establish the tractability of UCQ entailment under $ICAR$-semantics.

Theorem 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A}}$ KB and let Q be a UCQ. Deciding whether $\mathcal{K} \models_{ICAR} Q$ is in PTIME with respect to data complexity.*

Proof. First, we prove that the ABox returned by *Compute-ICAR-Repair*$(\mathcal{K})$ is the $ICAR$-repair of $\mathcal{K}$. This follows from the following property: let $\alpha_1, \alpha_2 \in$ *clc*$(\langle \mathcal{T}, \mathcal{A} \rangle)$ and let $\langle \mathcal{T}, \{\alpha_1, \alpha_2\} \rangle$ be unsatisfiable. Then, let $\beta_1 \in \mathcal{A}$ be such that $\langle \mathcal{T}, \{\beta_1\} \rangle \models \alpha_1$, and let $\beta_2 \in \mathcal{A}$ be such $\langle \mathcal{T}, \{\beta_2\} \rangle \models \alpha_2$ (such facts β_1, β_2 always exist). Now, it is immediate to verify that $\langle \mathcal{T}, \{\beta_1, \beta_2\} \rangle$ is unsatisfiable. Moreover, by definition of *clc*$(\langle \mathcal{T}, \mathcal{A} \rangle)$ we have that (i) $\langle \mathcal{T}, \{\beta_1\} \rangle$ is satisfiable, and (ii) $\langle \mathcal{T}, \{\beta_2\} \rangle$ is satisfiable. Now, (i) immediately implies that there exists a CAR-repair $\mathcal{A}'$ of $\mathcal{K}$ that contains β_1, and hence α_1 since $\mathcal{A}'$ is deductively closed. Consequently, α_2 cannot belong to the $\mathcal{A}'$ (since $\langle \mathcal{T}, \mathcal{A}' \rangle$ must be satisfiable, and hence α_2 does not belong to the intersection of all the CAR-repairs of $\mathcal{K}$. In the same way, from (ii) we derive that α_1 does not belong to the intersection of all the CAR-repairs of $\mathcal{K}$.

Now, as shown in the proof of Theorem 4, *clc*$(\mathcal{K})$ can be computed in polynomial time, which implies that the algorithm *Compute-ICAR-Repair*$(\mathcal{K})$ runs in polynomial time. Moreover, by definition of $ICAR$-semantics, we have that, for every UCQ Q,

	AR semantics	CAR semantics	IAR semantics	$ICAR$ semantics
instance checking	coNP-complete	in PTIME	in PTIME	in PTIME
UCQ entailment	coNP-complete [10]	coNP-complete	in PTIME	in PTIME

Fig. 2. Data complexity of UCQ entailment over *DL-Lite*$_{\mathcal{A}}$ KBs under inconsistency-tolerant semantics

$\mathcal{K} \models_{ICAR} Q$ iff $\langle \mathcal{T}, \mathcal{A}' \rangle \models Q$ where $\mathcal{A}'$ is the $ICAR$-repair of $\mathcal{K}$. Hence, from tractability of UCQ entailment in *DL-Lite*$_{\mathcal{A}}$, the thesis follows. □

Finally, we consider the instance checking problem under CAR-semantics, and show that instance checking under CAR-semantics coincides with instance checking under the $ICAR$-semantics.

Lemma 2. *Let $\mathcal{K}$ be a DL-Lite$_{\mathcal{A}}$ KB, and let α be an ABox assertion. Then, $\mathcal{K} \models_{CAR} \alpha$ iff $\mathcal{K} \models_{ICAR} \alpha$.*

Proof. $\mathcal{K} \models_{CAR} \alpha$ if $\mathcal{K} \models_{ICAR} \alpha$ follows from the fact that the $ICAR$-semantics is a sound approximation of the CAR-semantics. As for the converse, since every CAR-repair is deductively closed, it follows that $\mathcal{K} \models_{CAR} \alpha$ iff α belongs to the intersection of all the CAR-repairs of $\langle \mathcal{T}, \mathcal{A} \rangle$, i.e., to the $ICAR$-repair of $\mathcal{K}$. □

The above property and Theorem 6 allow us to establish tractability of instance checking under the CAR-semantics.

Theorem 7. *Let $\mathcal{K}$ be a DL-Lite$_{\mathcal{A}}$ KB, and let α be an ABox assertion. Deciding whether $\mathcal{K} \models_{CAR} \alpha$ is in PTIME with respect to data complexity.*

We remark that the analogous of Lemma 2 does *not* hold for AR, because AR-repairs are not deductively closed. This is the reason why instance checking under AR-semantics is harder, as stated by Theorem 3. In Figure 2 we summarize the complexity results presented in this section.

5 Conclusions

We have presented an investigation on inconsistency-tolerant reasoning in DLs, with special attention to the *DL-Lite* family. The techniques we have illustrated assume that the TBox is consistent, and therefore consider the case of inconsistencies arising between the TBox and the ABox.

Our approach to inconsistency-tolerance is inspired by the work done on consistent query answering in databases. Indeed, the AR-semantics presented in Section 3 is the direct application of the notion of repair to DL knowledge bases. Motivated by the intractability of inconsistency-tolerant query answering under such semantics, we have investigated several variants of the AR-semantics, with the goal of finding a good compromise between expressive power and complexity of query answering.

Our work can proceed along different directions. One notable problem we aim at addressing is the design of new algorithms for inconsistency-tolerant query answering both under the IAR-semantics and the $ICAR$-semantics, based on the idea of rewriting the query into a FOL query to be evaluated directly over the inconsistent ABox. We would also like to study reasoning under inconsistency-tolerant semantics in Description Logics outside the DL-lite family.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
2. Bertossi, L., Hunter, A., Schaub, T. (eds.): Inconsistency Tolerance. LNCS, vol. 3300. Springer, Heidelberg (2005)
3. Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proc. of PODS 2003, pp. 260–271 (2003)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
5. Chomicki, J.: Consistent query answering: Five easy pieces. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 1–17. Springer, Heidelberg (2006)
6. Eiter, T., Gottlob, G.: On the complexity of propositional knowledge base revision, updates and counterfactuals. Artificial Intelligence 57, 227–270 (1992)
7. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In: Proc. of IJCAI 2007, pp. 399–404 (2007)
8. Haasa, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y.: A framework for handling inconsistency in changing ontologies. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 353–367. Springer, Heidelberg (2005)
9. Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with inconsistent ontologies. In: Proc. of IJCAI 2005, pp. 454–459 (2003)
10. Lembo, D., Ruzzi, M.: Consistent query answering over description logic ontologies. In: Proc. of RR 2007 (2007)
11. Ma, Y., Hitzler, P.: Paraconsistent reasoning for owl 2. In: Proc. of RR 2009, pp. 197–211 (2009)
12. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Proc. of WWW 2005, pp. 633–640 (2005)
13. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
14. Qi, G., Du, J.: Model-based revision operators for terminologies in description logics. In: Proc. of IJCAI 2009, pp. 891–897 (2009)
15. Winslett, M.: Updating Logical Databases. Cambridge University Press, Cambridge (1990)

Extending Paraconsistent $\mathcal{SROIQ}$

Frederick Maier[⋆]

Kno.e.sis Center, Wright State University

Abstract. The four-valued paraconsistent logic $\mathcal{SROIQ}4$, originally presented by Ma and Hitzler, is extended to incorporate additional elements of $\mathcal{SROIQ}$. It is shown that the modified logic is classically sound and that its embedding into classical $\mathcal{SROIQ}$ is consequence preserving. Furthermore, inserting special axioms into a $\mathcal{SROIQ}4$ knowledge base allows additional nontrivial conclusions to be drawn, without affecting paraconsistency. It is also shown that the interaction of nominals and cardinality restrictions prevents some $\mathcal{SROIQ}4$ knowledge bases from having models. For such knowledge bases, the logic remains explosive.

1 Introduction

Standard description logics (DLs) are essentially classical in nature. In each interpretation, every proposition is either true or false but never both, a *model* of a set K of propositions is an interpretation making each member true, and K *entails* a proposition $\mathcal{P}$ ($K \models \mathcal{P}$) *iff* every model of K is a model of $\mathcal{P}$.

As a result, standard description logics obey the *principle of explosion*.

$$\text{If } K \models \mathcal{P} \text{ and } K \models \neg\mathcal{P}, \text{ then } K \models \mathcal{Q} \text{ for all } \mathcal{Q}.$$

If both a proposition $\mathcal{P}$ and its negation $\neg\mathcal{P}$ are entailed by a set of premises K, then all propositions $\mathcal{Q}$ are also entailed. K is *inconsistent* if it has no models, and if K has no models, then trivially all of them are models of all propositions. Furthermore, given the semantics of negation, it is impossible for both $\mathcal{P}$ and $\neg\mathcal{P}$ to be simultaneously true. And so the following are all equivalent in a classical logic: K is inconsistent; $K \models \mathcal{P}$ and $K \models \neg\mathcal{P}$; $K \models \mathcal{Q}$ for all $\mathcal{Q}$.

The unfortunate effect of this is that classical logic cannot be used as a guide to reasoning over inconsistent knowledge bases, since if everything can be inferred from an inconsistency, nothing useful can be. This is a significant problem, as inconsistencies are commonplace in real-world systems, and it is well known that determining consistency in expressive formal systems is quite difficult (sometimes impossible). And even if one can find inconsistencies, restoring consistency by deleting some subset of statements necessarily removes valuable information.

An alternative solution to the problem is to reject the principle of explosion itself, employing a *paraconsistent logic*. This is the approach taken by Yue Ma,

[⋆] This work was performed in part while the author was a postdoctoral fellow at the Florida Institute for Human and Machine Cognition, and later with support from the Wright State University Research Council.

Pascal Hitzler, Zuoquan Lin, and others in a collection of papers [9,10,11,12]. They present 4-valued paraconsistent semantics for selected DLs and show how knowledge bases under the new semantics can be translated into corresponding knowledge bases under classical DL semantics. The approach's great virtue is that it permits classical reasoners to draw inferences under the new semantics.

The present paper continues this work. We present a variation of the paraconsistent logic $\mathcal{SROIQ}4$ [9], adding further characteristic elements of classical $\mathcal{SROIQ}$. Specifically, we add (ir)reflexive and disjoint roles, a universal role, negative role assertions, role chains, and $R.Self$ concept descriptions. We show that the embedding of the extended logic into classical $\mathcal{SROIQ}$ is consequence preserving, and that inserting special axioms into a $\mathcal{SROIQ}4$ knowledge base can selectively enforce the law of the excluded middle (LEM) and the law of non-contradiction (LNC). This allows one to draw additional classically correct conclusions. Importantly, we also point out a limitation, hitherto unnoticed, of the paraconsistent framework of which $\mathcal{SROIQ}4$ forms a part. Namely, while many DL constructs do not affect paraconsistency, the interaction of nominals and cardinality restrictions prevents some $\mathcal{SROIQ}4$ knowledge bases from having four-valued models. For these, the principle of explosion still applies.

Section 2 below reviews multivalued logics and presents the operators forming the basis for concept inclusion in paraconsistent DLs. $\mathcal{SROIQ}4$ is defined in Section 3 and it is proved sound *wrt* classical $\mathcal{SROIQ}$. Section 4 shows how to eliminate truth-value gaps and gluts. The consequence preserving embedding of $\mathcal{SROIQ}4$ into $\mathcal{SROIQ}$ is given in Section 5. Section 6 shows how the interaction of nominals and cardinality restrictions causes paraconsistency to break down. A brief concluding section discusses related work.

2 Multivalued Paraconsistent Logics

A logic is paraconsistent if the principle of explosion fails in at least one case. In the 4-valued logics upon which $\mathcal{SROIQ}4$ is based, paraconsistency is achieved essentially by redefining what "model" means, so that classically inconsistent sets can have them. The use of four truth values $\{n, 0, 1, b\}$ can be traced to Belnap [4,5], with the values indicating what a computer has been told about $\mathcal{P}$: nothing (n), that $\mathcal{P}$ is false (0), that $\mathcal{P}$ is true (1), that $\mathcal{P}$ is true and that $\mathcal{P}$ is false (b). The value b is a truth value *glut*, while n is a truth value *gap*. The semantics for $\neg$ (*not*), $\wedge$ (*and*), $\vee$ (*or*) are defined using the below truth tables. Material implication $\mathcal{P} \mapsto \mathcal{Q}$ is taken as shorthand for $\neg\mathcal{P} \vee \mathcal{Q}$.

$\neg$	
n	n
0	1
1	0
b	b

$\wedge$	n	0	1	b
n	n	0	n	0
0	0	0	0	0
1	n	0	1	b
b	0	0	b	b

$\vee$	n	0	1	b
n	n	n	1	1
0	n	0	1	b
1	1	1	1	1
b	1	b	1	b

$\mapsto$	n	0	1	b
n	n	n	1	1
0	1	1	1	1
1	n	0	1	b
b	1	b	1	b

An interpretation $\mathcal{I}$ is a 4-model of a set K of formulas if each formula is assigned one of the *designated* values 1 or b. K entails $\mathcal{P}$ *iff* each 4-model of K

is a 4-model of $\mathcal{P}$. This straightforward extension of classical entailment causes several rules of inference and theorems to fail in the 4-valued context.

- **Disjunctive Syllogism (DS):** If $K \models \neg\mathcal{P}$ and $K \models \mathcal{P} \vee \mathcal{Q}$, then $K \models \mathcal{Q}$.
- **Modus Ponens (MP):** If $K \models \mathcal{P} \mapsto \mathcal{Q}$ and $K \models \mathcal{P}$, then $K \models \mathcal{Q}$.
- **Modus Tollens (MT):** If $K \models \mathcal{P} \mapsto \mathcal{Q}$ and $K \models \neg\mathcal{Q}$, then $K \models \neg\mathcal{P}$.
- **Deduction Theorem (DT):** If $K \cup \{\mathcal{P}\} \models \mathcal{Q}$, then $K \models \mathcal{P} \mapsto \mathcal{Q}$.

The combined failures of MP and the DT have led to the development of alternative implication operators, and $\mathcal{SROIQ}4$ uses operators defined in the logics of Arieli and Avron [2,3]. The operator ($\supset$) is called *internal implication*, and it forms the basis for *strong implication* ($\rightarrow$) and equivalence ($\leftrightarrow$). For any truth-value assignment v, the semantics for the three are given below.

- $v(\mathcal{P} \supset \mathcal{Q})=1$ if $v(\mathcal{P}) \notin \{1, b\}$; otherwise $v(\mathcal{P} \supset \mathcal{Q})=v(\mathcal{Q})$.
- $v(\mathcal{P} \rightarrow \mathcal{Q})=v((\mathcal{P} \supset \mathcal{Q}) \wedge (\neg\mathcal{Q} \supset \neg\mathcal{P}))$
- $v(\mathcal{P} \leftrightarrow \mathcal{Q}) = v((\mathcal{P} \rightarrow \mathcal{Q}) \wedge (\mathcal{Q} \rightarrow \mathcal{P}))$

Internal implication satisfies MP and DT, but *not* MT. Strong implication satisfies MP and MT but not DT. With equivalence, $\mathcal{P} \leftrightarrow \mathcal{Q}$ is designated *iff* the values of $\mathcal{P}$ and $\mathcal{Q}$ coincide. The analogous claim does *not* hold if equivalence is spelled out as $(\mathcal{P} \supset \mathcal{Q}) \wedge (\mathcal{Q} \supset \mathcal{P})$.

3 $\mathcal{SROIQ}$ and $\mathcal{SROIQ}4$

$\mathcal{SROIQ}4$ was first presented in [9], but we have modified it here by adding the features mentioned in the introduction. Well-formed formulas are created from the disjoint sets N_C, N_R, and N_I of *atomic concept names*, *atomic role names*, and *individual names*, together with a set N_o of nominals $\{a_1, \ldots, a_n\}$ (each $a_i \in N_I$), connectives $\neg$, $\sqcap$, $\sqcup$, etc., and punctuation marks. A *knowledge base* is a union of a TBox, RBox, and ABox, where a TBox is a set of concept inclusion axioms (GCIs), an RBox is a set of role inclusion axioms (RIAs) and role assertions, and an ABox is a set of individual concept and role assertions. A more detailed description of $\mathcal{SROIQ}$ can be found in [8].

$\mathcal{SROIQ}4$ allows GCIs based on the operators of Arieli and Avron: $(C_1 \mapsto C_2)$, *material inclusion*; $(C_1 \sqsubset C_2)$, *internal inclusion*; and $(C_1 \rightarrow C_2)$, *strong inclusion.* The new axioms are not formally defined in classical $\mathcal{SROIQ}$. Nevertheless, we choose to allow them in $\mathcal{SROIQ}$, stipulating that they have the same semantics as $\sqsubseteq$. We also allow $(C \sqsubseteq D)$ in $\mathcal{SROIQ}4$, reading it as $(C \sqsubset D)$. This allows a common language $\mathcal{L}$ for both $\mathcal{SROIQ}$ and $\mathcal{SROIQ}4$.

The semantics of $\mathcal{SROIQ}4$ assigns to each concept description both a *positive* extension P and a *negative* extension N. $C(a)$ can thereby take on one of Belnap's four values in an interpretation $\mathcal{I}$: 1 ($a^\mathcal{I} \in P$, $a^\mathcal{I} \notin N$); 0 ($a^\mathcal{I} \notin P$, $a^\mathcal{I} \in N$); n ($a^\mathcal{I} \notin P$, $a^\mathcal{I} \notin N$); b ($a^\mathcal{I} \in P$, $a^\mathcal{I} \in N$). We also give roles a 4-valued semantics:[1] $R^\mathcal{I}$ is $\langle P, N \rangle$, where P and N are subsets of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$. This maintains paraconsistency in the presence of, e.g., negative role assertions.

[1] In the earlier paper by Ma and Hitzler [9], roles are treated classically.

Table 1. Syntax and semantics of $\mathcal{SROIQ}4$ concept descriptions

Syntax	$\mathcal{SROIQ}4$ **Semantics**
$C\ (C \in N_C)$	$C^{\mathcal{I}} = \langle P, N \rangle$, where $P, N \subseteq \Delta^{\mathcal{I}}$
$\{a_1, \ldots, a_n\}\ (a_i \in N_I)$	$\langle \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}, N \rangle$, where $N \subseteq \Delta^{\mathcal{I}}$
$R\ (R \in N_R)$	$R^{\mathcal{I}} = \langle P, N \rangle$, where $P, N \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
R^-	$R^{-\mathcal{I}} = \langle P^-, N^- \rangle$, where $R^{\mathcal{I}} = \langle P, N \rangle$, $P^- = \{(b,a) \mid (a,b) \in P\}$, $N^- = \{(b,a) \mid (a,b) \in N\}$
U (Top Role)	$U^{\mathcal{I}} = \langle \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}, \varnothing \rangle$
$R_1 \circ \ldots \circ R_n$	$\langle \{(x,z) \mid (\exists y_{1 \ldots n})[(x,y_1) \in p^+(R_1^{\mathcal{I}}) \wedge \ldots \wedge (y_n,z) \in p^+(R_n^{\mathcal{I}}]\}) \{(x,z) \mid (\forall y_{1 \ldots n})[(x,y_1) \in p^-(R_1^{\mathcal{I}}) \vee \ldots \vee (y_n,z) \in p^-(R_n^{\mathcal{I}}]\}) \rangle$
$\top$	$\top^{\mathcal{I}} = \langle \Delta^{\mathcal{I}}, \varnothing \rangle$
$\bot$	$\bot^{\mathcal{I}} = \langle \varnothing, \Delta^{\mathcal{I}} \rangle$
$\neg C$	$(\neg C)^{\mathcal{I}} = \langle N, P \rangle$, where $C^{\mathcal{I}} = \langle P, N \rangle$
$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = \langle P_1 \cap P_2, N_1 \cup N_2 \rangle$, where $C_i^{\mathcal{I}} = \langle P_i, N_i \rangle$
$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = \langle P_1 \cup P_2, N_1 \cap N_2 \rangle$, where $C_i^{\mathcal{I}} = \langle P_i, N_i \rangle$
$\exists R.C$	$\langle \{x \mid (\exists y)[(x,y) \in p^+(R^{\mathcal{I}}) \wedge y \in p^+(C^{\mathcal{I}})]\}$, $\{x \mid (\forall y)[(x,y) \in p^+(R^{\mathcal{I}}) \mapsto y \in p^-(C^{\mathcal{I}})]\} \rangle$
$\forall R.C$	$\langle \{x \mid (\forall y)([(x,y) \in p^+(R^{\mathcal{I}}) \mapsto y \in p^+(C^{\mathcal{I}})]\}$, $\{x \mid (\exists y)[(x,y) \in p^+(R^{\mathcal{I}}) \wedge y \in p^-(C^{\mathcal{I}})]\} \rangle$
$\exists R.Self$	$\langle \{x \mid (x,x) \in p^+(R^{\mathcal{I}})\}, N \rangle$, $N \subseteq \Delta^{\mathcal{I}}$
$\leq nR.C$	$\langle \{x \mid \sharp\{y \mid (x,y) \in p^+(R^{\mathcal{I}}) \wedge y \notin p^-(C^{\mathcal{I}})\} \leq n\}$, $\{x \mid \sharp\{y \mid (x,y) \in p^+(R^{\mathcal{I}}) \wedge y \in p^+(C^{\mathcal{I}})\} > n\} \rangle$
$\geq nR.C$	$\langle \{x \mid \sharp\{y \mid (x,y) \in p^+(R^{\mathcal{I}}) \wedge y \in p^+(C^{\mathcal{I}})\} \geq n\}$, $\{x \mid \sharp\{y \mid (x,y) \in p^+(R^{\mathcal{I}}) \wedge y \notin p^-(C^{\mathcal{I}})\} < n\} \rangle$

Table 2. Syntax and semantics of $\mathcal{SROIQ}4$ axioms and assertions

Syntax	$\mathcal{SROIQ}4$ **Semantics**
$C_1 \mapsto C_2$	$(\Delta^{\mathcal{I}} - p^-(C_1^{\mathcal{I}})) \subseteq p^+(C_2^{\mathcal{I}})$
$C_1 \sqsubset C_2$	$p^+(C_1^{\mathcal{I}}) \subseteq p^+(C_2^{\mathcal{I}})$
$C_1 \to C_2$	$p^+(C_1^{\mathcal{I}}) \subseteq p^+(C_2^{\mathcal{I}})$ and $p^-(C_2^{\mathcal{I}}) \subseteq p^-(C_1^{\mathcal{I}})$
$R_1 \circ \ldots \circ R_n \sqsubseteq R_{n+1}$	$p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}}) \subseteq p^+(R_{n+1}^{\mathcal{I}})$
$Ref(R)$	$\{(x,x) \mid x \in \Delta^{\mathcal{I}}\} \subseteq p^+(R^{\mathcal{I}})$
$Irr(R)$	$\{(x,x) \mid x \in \Delta^{\mathcal{I}}\} \subseteq p^-(R^{\mathcal{I}})$
$Dis(R,S)$	$p^+(R^{\mathcal{I}}) \subseteq p^-(S^{\mathcal{I}})$ and $p^+(S^{\mathcal{I}}) \subseteq p^-(R^{\mathcal{I}})$
$C(a)$	$a^{\mathcal{I}} \in p^+(C^{\mathcal{I}})$
$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
$R(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^+(R^{\mathcal{I}})$
$\neg R(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^-(R^{\mathcal{I}})$

A *4-valued interpretation* (*4-interpretation*) is a tuple $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is nonempty and $\cdot^{\mathcal{I}}$ is a valuation function defined inductively as shown in Tables 1 and 2. If C is a concept and $\mathcal{I}$ is a 4-interpretation such that $C^{\mathcal{I}} = \langle P, N \rangle$, then $p^+(C^{\mathcal{I}}) =_{def} P$ and $p^-(C^{\mathcal{I}}) =_{def} N$. $\mathcal{I}$ 4-*satisfies* (is a 4-*model of*) axiom A if the corresponding condition of $\mathcal{SROIQ}4$ in Table 2 is met. $\mathcal{I}$ 4-satisfies a knowledge base KB *iff* it 4-satisfies each axiom in KB. KB is 4-*satisfiable* (4-*unsatisfiable*) *iff* it has (does not have) a 4-model. KB *entails* an axiom A *wrt* $\mathcal{SROIQ}4$ ($KB \models_{\mathcal{SROIQ}4} A$) *iff* every 4-model of KB is a 4-model of A.

Every 2-interpretation $\mathcal{I}_2$ of a knowledge base corresponds to a 4-interpretation $\mathcal{I}_4$ (the *4-counterpart* of $\mathcal{I}_2$): $\Delta^{\mathcal{I}_4} =_{def} \Delta^{\mathcal{I}_2}$; for each $a \in N_I$, $a^{\mathcal{I}_4} =_{def} a^{\mathcal{I}_2}$; for each $R \in N_R$, $R^{\mathcal{I}_4} =_{def} \langle R^{\mathcal{I}_2}, (\Delta^{\mathcal{I}_2} \times \Delta^{\mathcal{I}_2}) - R^{\mathcal{I}_2}\rangle$; for each $C \in N_C \cup N_o$, or $C = \exists R.Self$, $C^{\mathcal{I}_4} =_{def} \langle C^{\mathcal{I}_2}, \Delta^{\mathcal{I}_2} - C^{\mathcal{I}_2}\rangle$. Since $\mathcal{I}_4$ and $\mathcal{I}_2$ share a domain of discourse, we will simply write Δ (omitting the superscripts).

The following propositions show that $\mathcal{SROIQ}4$ is sound *wrt* $\mathcal{SROIQ}$.

Proposition 1. *If $\mathcal{I}_2$ is a 2-interpretation and $\mathcal{I}_4$ its 4-counterpart, then for any concept description C, $C^{\mathcal{I}_4} = \langle C^{\mathcal{I}_2}, \Delta - C^{\mathcal{I}_2}\rangle$.*

Proof. We induct on the degree of C. If $C \in N_C \cup N_o$ or $C = \exists R.Self$, the claim holds by definition. For $\top$, $p^+(\top^{\mathcal{I}_4}) = \Delta = \top^{\mathcal{I}_2}$ and $p^-(\top^{\mathcal{I}_4}) = \varnothing = \Delta - \Delta = \Delta - \top^{\mathcal{I}_2}$. For bottom, $p^+(\bot^{\mathcal{I}_4}) = \varnothing = \bot^{\mathcal{I}_2}$ and $p^-(\bot^{\mathcal{I}_4}) = \Delta = \Delta - \varnothing = \Delta - \bot^{\mathcal{I}_2}$. For the inductive cases, suppose the claim holds for all concepts of degree $< n$ and that C has degree n. We consider the cases where C is one of $\neg D$, $(D \sqcap E)$, $(\exists R.D)$, or $(\geq nR.D)$. The remaining cases are similar.

- $(\neg\mathbf{D})^{\mathcal{I}_4} = \langle p^-(D^{\mathcal{I}_4}), p^+(D^{\mathcal{I}_4})\rangle = \langle \Delta - D^{\mathcal{I}_2}, D^{\mathcal{I}_2})\rangle = \langle (\neg D)^{\mathcal{I}_2}, \Delta - (\neg D)^{\mathcal{I}_2}\rangle$.
- $(\mathbf{D} \sqcap \mathbf{E})^{\mathcal{I}_4} = \langle (p^+(D^{\mathcal{I}_4}) \cap p^+(E^{\mathcal{I}_4})), (p^-(D^{\mathcal{I}_4}) \cup p^-(E^{\mathcal{I}_4}))\rangle = \langle (D^{\mathcal{I}_2} \cap E^{\mathcal{I}_2}), ((\Delta - D^{\mathcal{I}_2}) \cup (\Delta - E^{\mathcal{I}_2}))\rangle = \langle (D^{\mathcal{I}_2} \cap E^{\mathcal{I}_2}), (\Delta - (D^{\mathcal{I}_2} \cap E^{\mathcal{I}_2}))\rangle$.
- $(\exists\mathbf{R}.\mathbf{D})^{\mathcal{I}_4} = \langle P, N\rangle$, where $P = \{x|(\exists y)[(x,y) \in p^+(R^{\mathcal{I}_4}) \wedge y \in p^+(D^{\mathcal{I}_4})]\} = \{x|(\exists y)[(x,y) \in R^{\mathcal{I}_2} \wedge y \in D^{\mathcal{I}_2}]\} = (\exists R.D)^{\mathcal{I}_2}$; and $N = \{x|(\forall y)[(x,y) \in p^+(R^{\mathcal{I}_4}) \mapsto y \in p^-(D^{\mathcal{I}_4})]\} = \{x|(\forall y)[(x,y) \in R^{\mathcal{I}_2} \mapsto y \in (\Delta - D^{\mathcal{I}_2})]\} = \{x|(\forall y)[(x,y) \in R^{\mathcal{I}_2} \mapsto y \notin D^{\mathcal{I}_2}]\} = \{x|\neg\neg(\forall y)[(x,y) \in R^{\mathcal{I}_2} \mapsto y \notin D^{\mathcal{I}_2}]\} = \{x|\neg(\exists y)[(x,y) \in R^{\mathcal{I}_2} \wedge y \in D^{\mathcal{I}_2}]\} = \Delta - (\exists R.D)^{\mathcal{I}_2}$.
- $(\geq \mathbf{nR}.\mathbf{D})^{\mathcal{I}_4} = \langle P, N\rangle$, where $P = \{x|\sharp\{y|(x,y) \in p^+(R^{\mathcal{I}_4}) \wedge y \in p^+(D^{\mathcal{I}_4})\} \geq n\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}_2} \wedge y \in D^{\mathcal{I}_2}\} \geq n\} = D^{\mathcal{I}_2}$; and $N = \{x|\sharp\{y|(x,y) \in p^+(R^{\mathcal{I}_4}) \wedge y \notin p^-(D^{\mathcal{I}_4})\} < n\} = \{x|\sharp\{y|(x,y) \in p^+(R^{\mathcal{I}_4}) \wedge y \in p^+(D^{\mathcal{I}_4})\} < n\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}_2} \wedge y \in D^{\mathcal{I}_2}\} < n\} = \Delta - (\geq nR.D)^{\mathcal{I}_2}$.

The below two lemmas are needed to prove Proposition 2, which relates the 2-models of an axiom A to its 4-models.

Lemma 1. *Let $\mathcal{I}_2$ be a 2-interpretation and $\mathcal{I}_4$ its 4-counterpart. If $R \in N_R$, then $R^{-\mathcal{I}_4} = \langle R^{-\mathcal{I}_2}, \Delta^2 - R^{-\mathcal{I}_2}\rangle$.*

Proof. By definition of $\mathcal{I}_4$, $R^{\mathcal{I}_4} = \langle R^{\mathcal{I}_2}, \Delta^2 - R^{\mathcal{I}_2}\rangle$. From this, $R^{-\mathcal{I}_4} = \langle R^{-\mathcal{I}_2}, (\Delta^2 - R^{\mathcal{I}_2})^-\rangle$. Observe that $(\Delta^2 - R^{\mathcal{I}_2})^- = (\Delta^2)^- - (R^{\mathcal{I}_2})^- = \Delta^2 - R^{-\mathcal{I}_2}$. And so $R^{-\mathcal{I}_4} = \langle R^{-\mathcal{I}_2}, \Delta^2 - R^{-\mathcal{I}_2})\rangle$.

Lemma 2. *If $\mathcal{I}_2$ is a 2-interpretation and $\mathcal{I}_4$ its 4-counterpart, then for roles $R_1, \ldots, R_n$, $(R_1 \circ \ldots \circ R_n)^{\mathcal{I}_4} = \langle (R_1 \circ \ldots \circ R_n)^{\mathcal{I}_2}, \Delta^2 - (R_1 \circ \ldots \circ R_n)^{\mathcal{I}_2}\rangle$.*

Proof. That $p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}_4}) = (R_1 \circ \ldots \circ R_n)^{\mathcal{I}_2}$ and $p^-((R_1 \circ \ldots \circ R_n)^{\mathcal{I}_4}) = \Delta^2 - (R_1 \circ \ldots \circ R_n)^{\mathcal{I}_2}$ follows by definition of $\mathcal{I}_4$ for roles and Lemma 1.

Proposition 2. *If $\mathcal{I}_2$ is a 2-interpretation and $\mathcal{I}_4$ its 4-counterpart, then for any axiom A, $\mathcal{I}_4$ is a 4-model of A* iff *$\mathcal{I}_2$ is a 2-model of A.*

Proof. (By cases):

- $\mathbf{A} = \mathbf{C}(\mathbf{a})$: $\mathcal{I}_4$ is a 4-model of $C(a)$ *iff* $a^{\mathcal{I}_4} \in p^+(C^{\mathcal{I}_4})$ *iff* $a^{\mathcal{I}_2} \in C^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $C(a)$.
- $\mathbf{A} = \mathbf{R}(\mathbf{a},\mathbf{b})$: $\mathcal{I}_4$ is a 4-model of $R(a,b)$ *iff* $(a^{\mathcal{I}_4}, b^{\mathcal{I}_4}) \in p^+(R^{\mathcal{I}_4})$ *iff* $(a^{\mathcal{I}_2}, b^{\mathcal{I}_2}) \in R^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $R(a,b)$.
- $\mathbf{A} = \neg\mathbf{R}(\mathbf{a},\mathbf{b})$: $\mathcal{I}_4$ is a 4-model of $\neg R(a,b)$ *iff* $(a^{\mathcal{I}_4}, b^{\mathcal{I}_4}) \in p^-(R^{\mathcal{I}_4})$ *iff* $(a^{\mathcal{I}_2}, b^{\mathcal{I}_2}) \notin R^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $\neg R(a,b)$.
- $\mathbf{A} = \mathbf{C} \sqsubset \mathbf{D}$: $\mathcal{I}_4$ is a 4-model of $(C \sqsubset D)$ *iff* $p^+(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ *iff* $C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $C \sqsubset D$.
- $\mathbf{A} = \mathbf{C} \mapsto \mathbf{D}$: $\mathcal{I}_4$ is a 4-model of $(C \mapsto D)$ *iff* $\Delta - p^-(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ *iff* $p^+(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ *iff* $C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $C \mapsto D$.
- $\mathbf{A} = \mathbf{C} \to \mathbf{D}$: $\mathcal{I}_4$ is a 4-model of $(C \to D)$ *iff* $(p^+(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ and $p^-(D^{\mathcal{I}_4}) \subseteq p^-(C^{\mathcal{I}_4}))$ *iff* $(C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$ and $\Delta - p^-(C^{\mathcal{I}_4}) \subseteq \Delta - p^-(D^{\mathcal{I}_4}))$ *iff* $(C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$ and $C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2})$ *iff* $\mathcal{I}_2$ is a 2-model of $C \to D$.
- $\mathbf{A} = \mathbf{R_1} \circ \ldots \circ \mathbf{R_n} \sqsubset \mathbf{R_{n+1}}$: $\mathcal{I}_4$ is a 4-model of $R_1 \circ \ldots \circ R_n \sqsubset R_{n+1}$*iff* $p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}_4}) \subseteq p^+(R_{n+1}^{\mathcal{I}_4})$ *iff* $(R_1 \circ \ldots \circ R_n)^{\mathcal{I}_2} \subseteq R_{n+1}^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $R_1 \circ \ldots \circ R_n \sqsubset R_{n+1}$.
- $\mathbf{A} = \mathbf{Ref}(\mathbf{R})$: $\mathcal{I}_4$ is a 4-model of $Ref(R)$ *iff* $\{(x,x) | x \in \Delta\} \subseteq p^+(R^{\mathcal{I}_4})$ *iff* $\{(x,x) | x \in \Delta\} \subseteq R^{\mathcal{I}_2}$ *iff* $\mathcal{I}_2$ is a 2-model of $Ref(R)$.
- $\mathbf{A} = \mathbf{Irr}(\mathbf{R})$: $\mathcal{I}_4$ is a 4-model of $Irr(R)$ *iff* $\{(x,x) | x \in \Delta\} \subseteq p^-(R^{\mathcal{I}_4})$ *iff* $\{(x,x) | x \in \Delta\} \subseteq (\Delta \times \Delta) - R^{\mathcal{I}_2}$ *iff* $\{(x,x) | x \in \Delta\} \cap R^{\mathcal{I}_2} = \varnothing$ *iff* $\mathcal{I}_2$ is a 2-model of $Irr(R)$.
- $\mathbf{A} = \mathbf{Dis}(\mathbf{R},\mathbf{S})$: $\mathcal{I}_4$ is a 4-model of $Dis(R,S)$ *iff* $(p^+(R^{\mathcal{I}_4}) \subseteq p^-(S^{\mathcal{I}_4})$ and $p^+(S^{\mathcal{I}_4}) \subseteq p^-(R^{\mathcal{I}_4}))$*iff* $(R^{\mathcal{I}_2} \subseteq \Delta^2 - S^{\mathcal{I}_2}$ and $S^{\mathcal{I}_2} \subseteq \Delta^2 - R^{\mathcal{I}_2})$ *iff* $R^{\mathcal{I}_2} \cap S^{\mathcal{I}_2} = \varnothing$ *iff* $\mathcal{I}_2$ is a 2-model of $Dis(R,S)$.

Since (in)equality assertions have the same semantics in both 2 -interpretations and 4–interpretations, we need show nothing for them.

Proposition 3 (Soundness of $\mathcal{SROIQ}4$). *Let KB be a $\mathcal{SROIQ}4$ knowledge base, A a $\mathcal{SROIQ}4$ axiom, and KB' and A' the knowledge base and axiom obtained by replacing each occurrence of $\sqsubset$, $\mapsto$, and $\to$ with $\sqsubseteq$.*

$$\textit{If } KB \models_{\mathcal{SROIQ}4} A, \textit{ then } KB' \models_{\mathcal{SROIQ}} A'$$

Proof. If $KB \models_{\mathcal{SROIQ}4} A$ and $\mathcal{I}_2$ 2-models KB', then by Prop. 2, the 4-counterpart $\mathcal{I}_4$ of $\mathcal{I}_2$ 4-models KB'. Again by Prop. 2, $\mathcal{I}_4$ 4-models KB and so A. By Prop. 2, $\mathcal{I}_2$ 2-models A. If $A \neq A'$, then A is a $\mathcal{SROIQ}4$ axiom, which has the same semantics in $\mathcal{SROIQ}$ as A'. And so $\mathcal{I}_2$ 2-models A'.

4 Removing Gaps and Gluts

A paraconsistent logic typically rejects one or both of the law of noncontradiction (LNC) $\neg(\mathcal{P} \wedge \neg\mathcal{P})$ and the law of the excluded middle (LEM) $\mathcal{P} \vee \neg\mathcal{P}$. $\mathcal{SROIQ}4$ rejects both. Nevertheless, adding further axioms to a knowledge base KB selectively enforces the laws, making KB "more classical." If LEM is enforced but not LNC, then additional classical consequences can be drawn while maintaining paraconsistency. Following Arieli [1], we define $EM(KB)$ ("excluded middle") and $EFQ(KB)$ ("ex falso quodlibet") in order to achieve this:

- $EM(KB) =_{def} \{\top \sqsubset (A \sqcup \neg A) : A = \exists R.Self$ or $A \in N_C \cup N_o\}$.
- $EFQ(KB) =_{def} \{(A \sqcap \neg A) \sqsubset \bot : A = \exists R.Self$ or $A \in N_C \cup N_o\}$.

Proposition 4. *$\mathcal{I}$ 4-models $EM(KB)$* iff *for each concept C of KB,*

$$p^+(C^\mathcal{I}) \cup p^-(C^\mathcal{I}) = \Delta^\mathcal{I}.$$

Proof. **(LR)** Let $\mathcal{I}$ 4-model $EM(KB)$. We induct on the degree of the concept C. For $C = \top$ and $C = \bot$, the claim holds by definition of $\top$ and $\bot$. If $C \in N_C \cup N_o$ or has the form $\exists R.Self$, since $\mathcal{I}$ 4-models $EM(KB)$, $\Delta^\mathcal{I} \subseteq p^+(C^\mathcal{I}) \cup p^+((\neg C)^\mathcal{I})$. Since $p^+(C^\mathcal{I}) \cup p^+((\neg C)^\mathcal{I}) \subseteq \Delta^\mathcal{I}$, the claim must hold. For the inductive cases, suppose the claim holds for concepts of degree $< n$ and that C has degree n. We consider selective cases. Proofs for the cases left out are analogous.

1. $\mathbf{C} = (\mathbf{D} \sqcap \mathbf{E})$: If $d \notin p^+(C^\mathcal{I})$, then $d \notin p^+(D)$ or $d \notin p^+(E)$. By ind. hyp., $d \in p^-(D)$ or $d \in p^-(E)$, and so $d \in p^-(D \sqcap E)$. I.e., $d \in p^-(C^\mathcal{I})$.
2. $\mathbf{C} = (\forall \mathbf{R}.\mathbf{D})$: If $d \notin p^+(C^\mathcal{I})$, then there exists a d' such that $(d, d') \in p^+(R^\mathcal{I})$ and $d' \notin p^+(D^\mathcal{I})$. But then by ind. hyp., $d' \in p^-(D^\mathcal{I})$, and so $d \in p^-(C^\mathcal{I})$.
3. $\mathbf{C} = (\leq \mathbf{nR}.\mathbf{D})$: If $d \notin p^+(C^\mathcal{I})$, then $\sharp\{y|(d, y) \in p^+(R^\mathcal{I}) \wedge y \notin p^-(D^{\mathcal{I}_4})\} > n\}$. But then by ind. hyp., $\sharp\{y|(d, y) \in p^+(R^\mathcal{I}) \wedge y \in p^+(D^{\mathcal{I}_4})\} > n\}$, and so $d \in p^-(C^\mathcal{I})$.

(RL) Suppose for each concept C, $p^+(C^\mathcal{I}) \cup p^-(C^\mathcal{I}) = \Delta^\mathcal{I}$. Then $p^+(C^\mathcal{I}) \cup p^+((\neg C)^\mathcal{I}) = \Delta^\mathcal{I}$, $\Delta \subseteq p^+(C^\mathcal{I}) \cup p^+((\neg C)^\mathcal{I})$, and so $p^+(\top^\mathcal{I}) \subseteq p^+((C \sqcup \neg C)^\mathcal{I})$. $\mathcal{I}$ is a 4-model of $\top \sqsubset (C \sqcup \neg C)$. Generalizing on C, $\mathcal{I}$ is a 4-model of $EM(KB)$.

Proposition 5. *$\mathcal{I}$ 4-models $EFQ(KB)$* iff *for each concept C of KB,*

$$p^+(C^\mathcal{I}) \cap p^-(C^\mathcal{I}) = \varnothing.$$

Proof. **(LR)** Suppose $\mathcal{I}$ 4-models $EFQ(KB)$. We induct on the degree of C. For $C = \top$ and $C = \bot$, the claim holds by definition. If $C \in N_C \cup N_o$ or has the form $\exists R.Self$, $(C \sqcap \neg C) \sqsubset \bot$ is satisfied, and so $p^+(C^\mathcal{I}) \cap p^+(\neg C)^\mathcal{I}) \subseteq \varnothing$. However, $\varnothing \subseteq p^+(C^\mathcal{I}) \cap p^+(\neg C)^\mathcal{I})$, and so $p^+(C^\mathcal{I}) \cap p^-(C^\mathcal{I}) = \varnothing$. For the inductive cases, suppose the claim holds for concepts of degree $< n$ and that C has degree n. We again consider selected cases.

1. If $d \in p^+((\mathbf{D} \sqcup \mathbf{E})^\mathcal{I})$, then $d \in p^+(D^\mathcal{I})$ or $d \in p^+(E^\mathcal{I})$. By ind. hyp., $d \notin p^-(D^\mathcal{I})$ or $d \notin p^-(E^\mathcal{I})$, and so $d \notin p^-((D \sqcup E)^\mathcal{I})$.
2. If $d \in p^+((\exists \mathbf{R}.\mathbf{D})^\mathcal{I})$, then there is a d' such that $(d, d') \in p^+(R^\mathcal{I})$ and $d' \in p^+(D^\mathcal{I})$. However, by ind. hyp., $d' \notin p^-(D^\mathcal{I})$. And so $d \notin p^-((\exists R.D)^\mathcal{I})$.
3. If $d \in p^-((\leq \mathbf{nR}.\mathbf{D})^\mathcal{I})$, then $\sharp\{y|(d, y) \in p^+(R^\mathcal{I}) \wedge y \in p^+(D^\mathcal{I})\} > n$. By ind. hyp., $\sharp\{y|(d, y) \in p^+(R^\mathcal{I}) \wedge y \notin p^-(D^\mathcal{I})\} > n$, and so $d \notin p^+((\leq nR.D)^\mathcal{I})$.

(RL) If $p^+(C^\mathcal{I}) \cap p^-(C^\mathcal{I}) = \varnothing$, then $p^+(C^\mathcal{I}) \cap p^+((\neg C)^\mathcal{I}) = \varnothing$, $p^+((C \sqcap \neg C)^\mathcal{I}) = \varnothing$, and so $(C \sqcap \neg C) \sqsubseteq \bot$ is satisfied by $\mathcal{I}$. Generalizing, $\mathcal{I}$ 4-models $EFQ(KB)$.

Adding $EM(KB)$ to KB brings $\mathcal{SROIQ}4$ closer to $\mathcal{SROIQ}$ without affecting paraconsistency. Furthermore, adding both $EM(KB)$ and $EFQ(KB)$ allows one to simulate—to a point—$\mathcal{SROIQ}$ reasoning in $\mathcal{SROIQ}4$. Let $\mathcal{I}_4$ be a 4-interpretation of KB. We define a corresponding 2-interpretation $\mathcal{I}_2$ as follows: $\Delta^{\mathcal{I}_2} =_{def} \Delta^{\mathcal{I}_4}$; for each individual $a \in N_I$, $a^{\mathcal{I}_2} =_{def} a^{\mathcal{I}_4}$; for each role $R \in N_R$, $R^{\mathcal{I}_2} =_{def} p^+(R^{\mathcal{I}_4})$; and for each $C \in N_C$, $C^{\mathcal{I}_2} =_{def} p^+(C^{\mathcal{I}_4})$.

Proposition 6. *$\mathcal{I}_4$ is a 4-model of $EM(KB) \cup EFQ(KB)$* iff *for each concept C, $C^{\mathcal{I}_4} = \langle C^{\mathcal{I}_2}, \Delta - C^{\mathcal{I}_2}\rangle$.*

Proof. **(LR)** Suppose $\mathcal{I}_4$ 4-models $EM(KB) \cup EFQ(KB)$. Let C be a concept of KB. In virtue of Props. 4 and 5, $\Delta - p^+(C^{\mathcal{I}_4}) = p^-(C^{\mathcal{I}_4})$. Inducting on the degree of C, we show that $C^{\mathcal{I}_4} = \langle C^{\mathcal{I}_2}, \Delta - C^{\mathcal{I}_2}\rangle$. We consider two cases below, both requiring that $\Delta - p^+(C^{\mathcal{I}_4}) = p^-(C^{\mathcal{I}_4})$. Other cases are similar.

1. $p^+((\neg\mathbf{C})^{\mathcal{I}_4}) = p^-(C^{\mathcal{I}_4}) = \Delta - C^{\mathcal{I}_2} = (\neg C)^{\mathcal{I}_2}$. So, $\Delta - (\neg C)^{\mathcal{I}_2} = p^-((\neg C)^{\mathcal{I}_4})$.
2. $p^+((\leq \mathbf{nR}.\mathbf{C})^{\mathcal{I}_4}) = \{x|\sharp\{y|(x,y) \in p^+(R^{\mathcal{I}_4}) \wedge y \notin p^-(C^{\mathcal{I}_4})\} \leq n\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}_2} \wedge y \in p^+(C^{\mathcal{I}_4})\} \leq n\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}_2} \wedge y \in C^{\mathcal{I}_2}\} \leq n\} = (\leq nR.C)^{\mathcal{I}_2}$. And so $\Delta - (\leq nR.C)^{\mathcal{I}_2} = p^-((\leq nR.C)^{\mathcal{I}_4})$.

(RL) If $C^{\mathcal{I}_4} = \langle C^{\mathcal{I}_2}, \Delta - C^{\mathcal{I}_2}\rangle$, then $\Delta \subseteq (p^+(C^{\mathcal{I}_4}) \cup p^+(\neg C^{\mathcal{I}_4}))$ and $(p^+(C^{\mathcal{I}_4}) \cap p^+((\neg C)^{\mathcal{I}_4})) \subseteq \varnothing$. As such, $\mathcal{I}_4$ 4-models $\top \sqsubset (C \sqcup \neg C)$ and $(C \sqcap \neg C) \sqsubset \bot$.

Proposition 7. *Let $\mathcal{I}_4$ 4-model $EM(KB) \cup EFQ(KB)$. For any axiom $A \in KB$ not of form $\neg R(a,b)$, $Irr(R)$, or $Dis(R,S)$, $\mathcal{I}_2$ 2-models A* iff *$\mathcal{I}_4$ 4-models A.*

Proof. We treat each case:

- $\mathcal{I}_4$ 4-models $(\mathbf{C} \sqsubset \mathbf{D})$ (or $\mathbf{C} \sqsubseteq \mathbf{D}$) *iff* $p^+(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ *iff* $C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$.
- $\mathcal{I}_4$ 4-models $(\mathbf{C} \mapsto \mathbf{D})$ *iff* $\Delta - p^-(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ *iff* $p^+(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ *iff* $C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$.
- $\mathcal{I}_4$ 4-models $(\mathbf{C} \rightarrow \mathbf{D})$ *iff* $(p^+(C^{\mathcal{I}_4}) \subseteq p^+(D^{\mathcal{I}_4})$ and $p^-(D^{\mathcal{I}_4}) \subseteq p^-(C^{\mathcal{I}_4}))$ *iff* $(C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$ and $\Delta - D^{\mathcal{I}_2} \subseteq \Delta - C^{\mathcal{I}_2})$ *iff* $C^{\mathcal{I}_2} \subseteq D^{\mathcal{I}_2}$.
- $\mathcal{I}_4$ 4-models $\mathbf{R_1} \circ \ldots \circ \mathbf{R_n} \sqsubset \mathbf{R_{n+1}}$ *iff* $p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}_4}) \subseteq p^+(R_{n+1}^{\mathcal{I}_4})$ *iff* $(R_1 \circ \ldots \circ R_n)^{\mathcal{I}_2} \subseteq R_{n+1}^{\mathcal{I}_2}$.
- $\mathcal{I}_4$ 4-models $\mathbf{Ref}(\mathbf{R})$ *iff* $\{(x,x)|x \in \Delta\} \subseteq p^+(R^{\mathcal{I}_4})$ *iff* $\{(x,x)|x \in \Delta\} \subseteq R^{\mathcal{I}_2}$.
- $\mathcal{I}_4$ 4-models $\mathbf{C}(\mathbf{a})$ *iff* $a^{\mathcal{I}_4} \in p^+(C^{\mathcal{I}_4})$ *iff* $a^{\mathcal{I}_2} \in C^{\mathcal{I}_2}$.
- $\mathcal{I}_4$ 4-models $\mathbf{R}(\mathbf{a},\mathbf{b})$ *iff* $(a^{\mathcal{I}_4}, b^{\mathcal{I}_4}) \in R^{\mathcal{I}_4}$ *iff* $(a^{\mathcal{I}_2}, b^{\mathcal{I}_2}) \in R^{\mathcal{I}_2}$.

Since (in)equalities are treated the same in the logics, we need not consider them.

Proposition 8. *If KB is a $\mathcal{SROIQ}4$ knowledge base lacking axioms of the form $\neg R(a,b)$, $Irr(R)$, or $Dis(R,S)$, then $KB \cup EM(KB) \cup EFQ(KB)$ has a 4-model* iff *KB has a 2-model.*

Proof. **(LR)** Let $\mathcal{I}_4$ 4-model $KB \cup EM(KB) \cup EFQ(KB)$. Prop. 7 applies, and so $\mathcal{I}_2$ (as described above) 2-models KB. **(RL)** Let $\mathcal{I}_2$ be any 2-model of KB. We may define a 4-counterpart $\mathcal{I}_4$ as in Section 3. By Proposition 2, for any axiom A of KB, $\mathcal{I}_2$ 2-models A *iff* $\mathcal{I}_4$ 4-models A. And so $\mathcal{I}_4$ 4-models KB. Clearly, in any 2-interpretation, every axiom of $EM(KB)$ and $EFQ(KB)$ is satisfied. Again by Proposition 2, $\mathcal{I}_4$ 4-models $EM(KB)$ and $EFQ(KB)$.

The mentioned role assertions are banned in Propositions 7 and 8 because counterparts to $EM(KB)$ and $EFQ(KB)$ for roles are not allowed in $\mathcal{SROIQ}$ or $\mathcal{SROIQ}4$. This ban can be lifted if roles are interpreted classically.

Table 3. Embedding $\mathcal{SROIQ}4$ axioms into $\mathcal{SROIQ}$

$\pi(C) = C$, where $C \in N_C$	$\pi(\neg C) = C'$, where C' new
$\pi(o) = o$, where $o \in N_o$	$\pi(\neg o) = C_o$, $o \in N_o$ and C_o new
	$\pi(\neg\neg C) = \pi(C)$
$\pi(\top) = \top$	$\pi(\neg\top) = \bot$
$\pi(\bot) = \bot$	$\pi(\neg\bot) = \top$
$\pi(E \sqcap D) = \pi(E) \sqcap \pi(D)$	$\pi(\neg(E \sqcap D)) = \pi(\neg E) \sqcup \pi(\neg D)$
$\pi(E \sqcup D) = \pi(E) \sqcup \pi(D)$	$\pi(\neg(E \sqcup D)) = \pi(\neg E) \sqcap \pi(\neg D)$
$\pi(\forall R.C) = \forall R.\pi(C)$	$\pi(\neg(\forall R.C)) = \exists R.\pi(\neg C)$
$\pi(\exists R.C) = \exists R.\pi(C)$	$\pi(\neg(\exists R.C)) = \forall R.\pi(\neg C)$
$\pi(\exists R.Self) = \exists R.Self$	$\pi(\neg\exists R.Self) = C_{R.Self}$
$\pi(\leq nR.C) = \leq nR.\neg\pi(\neg C)$	$\pi(\neg \leq nR.C) = \geq (n+1)R.\pi(C)$
$\pi(\geq nR.C) = \geq nR.\pi(C)$	$\pi(\neg \geq nR.C) = \leq (n-1)R.\neg\pi(\neg C)$
$\pi(C(a)) = \pi(C)(a)$	$\pi(a \neq b) = (a \neq b)$
$\pi(R(a,b)) = R(a,b)$	$\pi(\neg R(a,b)) = R'(a,b)$
$\pi(Ref(R)) = Ref(R)$	$\pi(Irr(R)) = Ref(R')$
$\pi(Dis(R,S)) = \{R \sqsubseteq S', S \sqsubseteq R'\}$	
$\pi(C \mapsto D) = \neg\pi(\neg C) \sqsubseteq \pi(D)$	*material inclusion*
$\pi(C \sqsubset D) = \pi(C) \sqsubseteq \pi(D)$	*internal inclusion*
$\pi(C \rightarrow D) = \{\pi(C \sqsubset D), \pi(\neg D \sqsubset \neg C)\}$	*strong inclusion*
$\pi(w \sqsubseteq R_{n+1}) = w \sqsubseteq R_{n+1}$, with $w = R_1 \circ \ldots \circ R_n$	*role inclusion*

5 Embedding $\mathcal{SROIQ}4$ into $\mathcal{SROIQ}$

To allow the use of classical DL reasoners with $\mathcal{SROIQ}4$, Ma et al. provide a translation function π (Table 3) from $\mathcal{SROIQ}4$ into $\mathcal{SROIQ}$. We modify it here. Each $C \in N_C$ is unaffected, but $\neg C$ becomes a new concept C'. In this way the 4-satisfiable $C \sqcap \neg C$ becomes the 2-satisfiable $C \sqcap C'$. Below, $\mathcal{L}$ refers to the language of $\mathcal{SROIQ}4$, and $\mathcal{L}'$ to the language created by adding primed counterparts to all atomic roles and concepts of $\mathcal{L}$. We also add new atomic concepts $C_{R.Self}$ for each role R in $\mathcal{L}$, and C_o for each nominal $o \in N_o$. These will be used to stand for the negations of $R.Self$ and nominals, respectively.

In general, if KB is a $\mathcal{SROIQ}4$ knowledge base, then $\pi(KB)$ is a $\mathcal{SROIQ}$ knowledge base. Observe that the usual relationship between $K \models \mathcal{P}$ and the unsatisfiability of $K \cup \{\neg\mathcal{P}\}$ does not hold in $\mathcal{SROIQ}4$. Nevertheless, if KB entails $\mathcal{P}$ in $\mathcal{SROIQ}4$, then $\pi(KB)$ entails $\pi(\mathcal{P})$ in $\mathcal{SROIQ}$ (Proposition 11). From this, it follows that $\pi(KB) \cup \{\neg\pi(\mathcal{P})\}$ is classically inconsistent.

If $\mathcal{I}$ is a 4-interpretation of $\mathcal{L}$, then define the 2-interpretation $\mathcal{I}'$ (the primed-counterpart of $\mathcal{I}$) over $\mathcal{L}'$ as follows:

- $\Delta^{\mathcal{I}'} =_{def} \Delta^{\mathcal{I}}$.
- For each individual $a \in N_I$, $a^{\mathcal{I}'} =_{def} a^{\mathcal{I}}$.
- For each role $R \in N_R \cup N_R^-$, $R^{\mathcal{I}'} =_{def} p^+(R^{\mathcal{I}})$ and $R'^{\mathcal{I}'} =_{def} p^-(R^{\mathcal{I}})$.
- For each atomic concept $C \in N_C$, $C^{\mathcal{I}'} =_{def} p^+(C^{\mathcal{I}})$ and $C'^{\mathcal{I}'} =_{def} p^-(C^{\mathcal{I}})$.
- For each nominal $o \in N_o$, $o^{\mathcal{I}'} =_{def} p^+(o^{\mathcal{I}})$, and $C_o^{\mathcal{I}'} =_{def} p^-(o^{\mathcal{I}})$.

– For each role $R \in N_R \cup N_R^-$, $(\exists R.Self)^{\mathcal{I}'} =_{def} p^+((\exists R.Self)^{\mathcal{I}})$, and
– For each role $R \in N_R \cup N_R^-$, $C_{R.Self}^{\mathcal{I}'} =_{def} p^-((\exists R.Self)^{\mathcal{I}})$.

It is clear that there is a 1–1 correspondence between the 4-interpretations of $\mathcal{L}$ and the 2-interpretations of $\mathcal{L}'$.

Proposition 9. *For any 4-interpretation $\mathcal{I}$, primed-counterpart $\mathcal{I}'$, and $\mathcal{SROIQ}4$ concept C, $p^+(C^{\mathcal{I}}) = \pi(C)^{\mathcal{I}'}$ and $p^-(C^{\mathcal{I}}) = \pi(\neg C)^{\mathcal{I}'}$ hold.*

Proof. We induct on the degree of C. If C is atomic, then $\pi(C) = C$, and by definition $p^+(C^{\mathcal{I}}) = C^{\mathcal{I}'} = \pi(C)^{\mathcal{I}'}$ and $p^-(C^{\mathcal{I}}) = C'^{\mathcal{I}'} = \pi(\neg C)^{\mathcal{I}'}$. If $C = \exists R.Self$, then $\pi(\exists R.Self) = \exists R.Self$, and by definition $(\exists R.Self)^{\mathcal{I}'} = p^+((\exists R.Self)^{\mathcal{I}})$. Furthermore, $\pi(\neg\exists R.Self) = C_{R.Self}$, and by definition $(C_{R.Self})^{\mathcal{I}'} = p^-((\exists R.Self)^{\mathcal{I}})$. If $o \in N_o$, then $\pi(o) = o$, and by definition $o^{\mathcal{I}'} = p^+(o^{\mathcal{I}})$. Furthermore, $\pi(\neg o) = C_o$, and by definition $(C_o)^{\mathcal{I}'} = p^-(o^{\mathcal{I}})$.

For the induction, we consider selected cases (the others are similar).

1. $p^+((\mathbf{D} \sqcup \mathbf{E})^{\mathcal{I}}) = p^+(D) \cup p^+(E) = \pi(D)^{\mathcal{I}'} \cup \pi(E)^{\mathcal{I}'} = (\pi(D) \sqcup \pi(E))^{\mathcal{I}'} = \pi(D \sqcup E)^{\mathcal{I}'}$.
   $p^-((\mathbf{D} \sqcup \mathbf{E})^{\mathcal{I}}) = p^-(D) \cap p^-(E) = \pi(\neg D)^{\mathcal{I}'} \cap \pi(\neg E)^{\mathcal{I}'} = (\pi(\neg D) \sqcap \pi(\neg E))^{\mathcal{I}'} = (\pi(\neg(D \sqcup E))^{\mathcal{I}'}$.
2. $p^+((\exists \mathbf{R}.\mathbf{D})^{\mathcal{I}}) = \{x|(\exists y)[(x,y) \in p^+(R^{\mathcal{I}}) \wedge y \in p^+(D^{\mathcal{I}})]\} = \{x|(\exists y)[(x,y) \in R^{\mathcal{I}'} \wedge y \in (\pi(D))^{\mathcal{I}'}]\} = (\exists R.\pi(D))^{\mathcal{I}'} = (\pi(\exists R.D))^{\mathcal{I}'}$.
   $p^-((\exists \mathbf{R}.\mathbf{D})^{\mathcal{I}}) = \{x|(\forall y)[(x,y) \in p^+(R^{\mathcal{I}}) \mapsto y \in p^-(D^{\mathcal{I}})]\} = \{x|(\forall y)[(x,y) \in R^{\mathcal{I}'} \mapsto y \in (\pi(\neg D))^{\mathcal{I}'}]\} = (\forall R.\pi(\neg D))^{\mathcal{I}'} = (\pi(\neg\exists R.D))^{\mathcal{I}'}$.
3. $p^+((\leq \mathbf{nR}.\mathbf{D})^{\mathcal{I}}) = \{x|\sharp\{y|(x,y) \in p^+(R^{\mathcal{I}}) \wedge y \notin p^-(D^{\mathcal{I}})\} \leq n\} =$
   $\{x|\sharp\{y|(x,y) \in R^{\mathcal{I}'} \wedge y \notin \pi(\neg D)^{\mathcal{I}'}\} \leq n\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}'} \wedge y \in (\Delta - \pi(\neg D)^{\mathcal{I}'})\} \leq n\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}'} \wedge y \in (\neg\pi(\neg D))^{\mathcal{I}'}\} \leq n\} = (\leq nR.\neg\pi(\neg D))^{\mathcal{I}'} = (\pi(\leq nR.D))^{\mathcal{I}'}$.
   $p^-((\leq \mathbf{nR}.\mathbf{D})^{\mathcal{I}}) = \{x|\sharp\{y|(x,y) \in p^+(R^{\mathcal{I}}) \wedge y \in p^+(D^{\mathcal{I}})\} \geq (n+1)\} = \{x|\sharp\{y|(x,y) \in R^{\mathcal{I}'} \wedge y \in \pi(D)^{\mathcal{I}'}\} \geq (n+1)\} = (\geq (n+1)R.\pi(D))^{\mathcal{I}'} = (\pi(\neg \leq nR.D))^{\mathcal{I}'}$.

   The remaining cases, involving negation, all have the same form:
4. $p^+((\neg D)^{\mathcal{I}}) = p^-(D^{\mathcal{I}}) = \pi((\neg D))^{\mathcal{I}'}$.
   $p^-((\neg D)^{\mathcal{I}}) = p^+(D^{\mathcal{I}}) = \pi(D)^{\mathcal{I}'} = \pi(\neg\neg D)^{\mathcal{I}'}$.

Lemma 3. *For any 4-interpretation $\mathcal{I}$ and primed-counterpart $\mathcal{I}'$, if S is the inverse of $R \in N_R$, then $p^+(S^{\mathcal{I}}) = S^{\mathcal{I}'}$.*

Proof. If $(a,b) \in p^+(S^{\mathcal{I}})$, then $(b,a) \in p^+(R^{\mathcal{I}})$. As such, $(b,a) \in R^{\mathcal{I}'}$, and so $(a,b) \in S^{\mathcal{I}'}$. If $(a,b) \in S^{\mathcal{I}'}$, then $(b,a) \in R^{\mathcal{I}'}$, and so both $(b,a) \in p^+(R^{\mathcal{I}})$ and $(a,b) \in p^+(S^{\mathcal{I}})$.

Lemma 4. *For any 4-interpretation $\mathcal{I}$ and primed-counterpart $\mathcal{I}'$,*

$$p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}}) = (R_1 \circ \ldots \circ R_n)^{\mathcal{I}'}.$$

Proof. **(LR)** Suppose $(x,z) \in p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}})$. Then there exist elements $y_1, \ldots, y_{n-1}$ such that $(x,y_1) \in p^+(R_1^{\mathcal{I}})$, $(y_1,y_2) \in p^+(R_2^{\mathcal{I}})$, …, $(y_{n-1},z) \in$

$p^+(R_n^{\mathcal{I}})$. Observe that for each R_i, $p^+(R_i^{\mathcal{I}}) = R_i^{\mathcal{I}'}$ by definition of $\mathcal{I}'$. And so $(x, y_1) \in R_1^{\mathcal{I}'}$, $(y_1, y_2) \in R_2^{\mathcal{I}'}$, …, $(y_{n-1}, z) \in R_n^{\mathcal{I}'}$. As such, $(x, z) \in (R_1 \circ \ldots \circ R_n)^{\mathcal{I}'}$. **(RL)** Suppose $(x, z) \in (R_1 \circ \ldots \circ R_n)^{\mathcal{I}'}$. Then there exist $y_1, \ldots, y_{n-1}$ such that $(x, y_1) \in R_1^{\mathcal{I}'}$, $(y_1, y_2) \in R_2^{\mathcal{I}'}$, …, $(y_{n-1}, z) \in R_n^{\mathcal{I}'}$. For each R_i, $p^+(R_i^{\mathcal{I}}) = R_i^{\mathcal{I}'}$ by definition of $\mathcal{I}'$. And so $(x, y_1) \in p^+(R_1^{\mathcal{I}})$, $(y_1, y_2) \in p^+(R_2^{\mathcal{I}})$, …, $(y_{n-1}, z) \in p^+(R_n^{\mathcal{I}})$. As such, $(x, z) \in p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}})$.

Proposition 10. *For any 4-interpretation $\mathcal{I}$, $\mathcal{I}$ is a 4-model of $\mathcal{SROIQ}4$ axiom A* iff *its primed-counterpart $\mathcal{I}'$ is a 2-model of $\pi(A)$.*

Proof. (By cases):

1. $\mathbf{C(a)}$: $\pi(C(a)) = \pi(C)(a)$. By Prop. 9, $a^{\mathcal{I}} \in p^+(C^{\mathcal{I}})$ *iff* $a^{\mathcal{I}'} \in \pi(C)^{\mathcal{I}'}$.
2. $\mathbf{A = R(a, b)}$. $\pi(R(a, b)) = R(a, b)$. Since $p^+(R^{\mathcal{I}}) = R^{\mathcal{I}'}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) = (a^{\mathcal{I}'}, b^{\mathcal{I}'})$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^+(R^{\mathcal{I}})$ *iff* $(a^{\mathcal{I}'}, b^{\mathcal{I}'}) \in R^{\mathcal{I}'}$.
3. $\mathbf{A = \neg R(a, b)}$. $\pi(\neg R(a, b)) = R'(a, b)$. Since $p^-(R^{\mathcal{I}}) = R'^{\mathcal{I}'}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) = (a^{\mathcal{I}'}, b^{\mathcal{I}'})$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in p^+((\neg R)^{\mathcal{I}})$ *iff* $(a^{\mathcal{I}'}, b^{\mathcal{I}'}) \in R'^{\mathcal{I}'}$.
4. $\mathbf{A = (C \sqsubset D)}$: $p^+(C^{\mathcal{I}}) \subseteq p^+(D^{\mathcal{I}})$ *iff* $\pi(C)^{\mathcal{I}'} \subseteq \pi(D)^{\mathcal{I}'}$ *iff* $\mathcal{I}'$ 2-models $(\pi(C) \sqsubset \pi(D))$ *iff* $\mathcal{I}'$ 2-models $\pi(C \sqsubset D)$.
5. $\mathbf{A = (C \rightarrow D)}$:
   $(p^+(C^{\mathcal{I}}) \subseteq p^+(D^{\mathcal{I}})$ and $p^-(D^{\mathcal{I}}) \subseteq p^-(C^{\mathcal{I}}))$ *iff* $(\pi(C)^{\mathcal{I}'} \subseteq \pi(D)^{\mathcal{I}'}$ and $\pi(\neg D)^{\mathcal{I}'} \subseteq \pi(\neg C)^{\mathcal{I}'})$ *iff* ($\mathcal{I}'$ 2-models $\{\pi(C) \sqsubseteq \pi(D),\ \pi(\neg D) \sqsubseteq \pi(\neg C)\}$ *iff* $\mathcal{I}'$ 2-models $\pi(C \rightarrow D)$.
6. $\mathbf{A = (C \mapsto D)}$: $(\Delta - p^-(C^{\mathcal{I}})) \subseteq p^+(D^{\mathcal{I}})$ *iff* $(\Delta - \pi(\neg C)^{\mathcal{I}'}) \subseteq \pi(D)^{\mathcal{I}'}$ *iff* $(\neg\pi(\neg C))^{\mathcal{I}'} \subseteq \pi(D)^{\mathcal{I}'}$ *iff* $\mathcal{I}'$ 2-models $(\neg\pi(\neg C) \sqsubseteq \pi(D))$ *iff* $\mathcal{I}'$ 2-models $\pi(C \mapsto D)$.
7. $\mathbf{R_1 \circ \ldots \circ R_n \sqsubset R_{n+1}}$: $\mathcal{I}$ 4-models $R_1 \circ \ldots \circ R_n \sqsubset R_{n+1}$ *iff*
   $p^+((R_1 \circ \ldots \circ R_n)^{\mathcal{I}}) \subseteq p^+(R_{n+1}^{\mathcal{I}})$ *iff* $(R_1 \circ \ldots \circ R_n)^{\mathcal{I}'} \subseteq R_{n+1}^{\mathcal{I}'}$ *iff* $\mathcal{I}'$ 2-models $R_1 \circ \ldots \circ R_n \sqsubseteq R_{n+1}$ *iff* $\mathcal{I}'$ 2-models $\pi(R_1 \circ \ldots \circ R_n \sqsubset R_{n+1})$.
8. $\mathbf{A = Ref(R)}$: $\mathcal{I}$ 4-models $Ref(R)$ *iff* $\{(x, x) | x \in \Delta\} \subseteq p^+(R^{\mathcal{I}})$ *iff*
   $\{(x, x) | x \in \Delta\} \subseteq R^{\mathcal{I}'}$ *iff* $\mathcal{I}'$ 2-models $Ref(R)$ *iff* $\mathcal{I}'$ 2-models $\pi(Ref(R))$.
9. $\mathbf{A = Irr(R)}$: $\mathcal{I}$ 4-models $Irr(R)$ *iff* $\{(x, x) | x \in \Delta\} \subseteq p^-(R^{\mathcal{I}})$ *iff* $\{(x, x) | x \in \Delta\} \subseteq R'^{\mathcal{I}'}$ *iff* $\mathcal{I}'$ 2-models $Ref(R')$ *iff* $\mathcal{I}'$ 2-models $\pi(Irr(R))$.
10. $\mathbf{A = Dis(R, S)}$: $\mathcal{I}$ 4-models $Dis(R, S)$ *iff* $(p^+(R^{\mathcal{I}}) \subseteq p^-(S^{\mathcal{I}})$ and $p^+(S^{\mathcal{I}}) \subseteq p^-(R^{\mathcal{I}}))$ *iff* $(R^{\mathcal{I}'} \subseteq S'^{\mathcal{I}'}$ and $S^{\mathcal{I}'} \subseteq R'^{\mathcal{I}'})$ *iff* $\mathcal{I}'$ 2-models $R \sqsubset S'$ and $S \sqsubset R'$ *iff* $\mathcal{I}'$ 2-models $\pi(Dis(R, S))$.

Proposition 11. *For any $\mathcal{SROIQ}4$ knowledge base KB and axiom A, $KB \models_{\mathcal{SROIQ}4} A$* iff *$\pi(KB) \models_{\mathcal{SROIQ}} \pi(A)$.*

Proof. **(LR)** Suppose $KB \models_{\mathcal{SROIQ}4} A$ and let $\mathcal{I}'$ 2-model $\pi(KB)$. Assume *wlog* that $\mathcal{I}'$ is the primed-counterpart of 4-interpretation $\mathcal{I}$ of KB. Since $\mathcal{I}'$ 2-models $\pi(KB)$, by Prop. 10 $\mathcal{I}$ 4-models KB and hence A. By Prop. 10, $\mathcal{I}'$ 2-models $\pi(A)$. Generalizing on $\mathcal{I}'$, $\pi(KB) \models_{\mathcal{SROIQ}} \pi(A)$. **(RL)** Suppose $\pi(KB) \models_{\mathcal{SROIQ}} \pi(A)$ and that $\mathcal{I}$ 4-models KB. Then there is a 2-interpretation $\mathcal{I}'$ that is the primed-counterpart of $\mathcal{I}$. By Prop. 10, since $\mathcal{I}$ 4-models KB, $\mathcal{I}'$ 2-models $\pi(KB)$ and so $\pi(A)$. By Prop 10, $\mathcal{I}$ 4-models A. Generalizing on $\mathcal{I}$, $KB \models_{\mathcal{SROIQ}4} A$.

6 Partial Paraconsistency

$\mathcal{SROIQ}4$ is intended to avoid the explosions caused by inconsistencies. Unfortunately, some $\mathcal{SROIQ}4$ knowledge bases—e.g., $\{(\top \sqcap \bot)(a)\}$—have no 4-models, and so $\mathcal{SROIQ}4$ is only partially paraconsistent. In some cases, inconsistent knowledge bases can be re-written into a classically equivalent form that avoids explosion. However, this is not the case in general. This was a point overlooked in the earlier work on $\mathcal{SROIQ}4$ [9], and we discuss it here.

Ma et al. report [12] that consistency can be maintained in $\mathcal{SHIQ}4$ by replacing $\top$ and $\bot$ with classically equivalent formulas. Specifically, let $SF(KB)$ be the knowledge base obtained by replacing each $\top$ with $A \sqcup \neg A$ and each $\bot$ with $A \sqcap \neg A$ (where A is a new atomic concept). $SF(KB)$ is guaranteed to possess a 4-model. However, nominals can conflict with cardinality restrictions, and so the analogous claim does not hold for $\mathcal{SROIQ}4$. If $|p^+(\{a_1, \dots, a_m\}^\mathcal{I})| = n$, then $\geq n+1 R.\{a_1, \dots, a_m\}(b)$ is not satisfiable.[2] And so, to maintain satisfiability, either cardinality restrictions or nominals must go.

Definition 12. *If C is a $\mathcal{SROIQ}4$ concept description and KB a $\mathcal{SROIQ}4$ knowledge base, then C is 4-satisfiable* iff *there is a 4-valued model $\mathcal{I}$ of KB such that $p^+(C^\mathcal{I}) \neq \varnothing$.*

Proposition 13. *If C is a concept description in which nominals, $\top$, and $\bot$ do not appear, then C is 4-satisfiable.*

Proof. Let n be the largest integer used in an $\geq nR.C$ restriction. Let $C^\mathcal{I} = \langle \Delta, \Delta \rangle$ for each $C \in N_C$, where $|\Delta| = n+1$. For each $R \in N_R$, let $R^\mathcal{I} = \langle \Delta^2, \Delta^2 \rangle$ and $(\exists R.Self)^\mathcal{I} = \langle \Delta, \Delta \rangle$. We induct on a the degree of C. If $C \in N_C$, the claim clearly holds. Examining the $\mathcal{SROIQ}4$ semantics shows that where C is $\neg D$, $D \sqcup E$, $D \sqcap E$, $\forall R.D$, $\exists R.D$, $\geq nR.D$, or $\leq nR.D$, $C^\mathcal{I} = \langle \Delta, \Delta \rangle$.

Proposition 14. *If KB is a $\mathcal{SROIQ}4$ knowledge base in which inequality assertions and nominals do not appear, then $SF(KB)$ has a 4-model.*

Proof. We use the interpretation $\mathcal{I}$ above. For simple assertions $C(a)$ and $R(a, b)$, the claim obviously holds. Since the anti-extension of each role R is Δ^2, $\neg R(a, b)$ also holds (we here reasonably assume that $R \neq U$), as do $Ref(R)$, $Irr(R)$, and $Dis(R, S)$. Since for all concepts C and D, $C^\mathcal{I} = D^\mathcal{I} = \langle \Delta, \Delta \rangle$, it follows that $C \sqsubset D$ and $C \rightarrow D$ hold. For $C \mapsto D$, observe that $\Delta^\mathcal{I} - p^-(C^\mathcal{I}) = \varnothing$, and so $C \mapsto D$ must be true in $\mathcal{I}$. The cases for RIAs are similar to $C \sqsubset D$.

Proposition 15. *If C is a $\mathcal{SROIQ}4$ concept description in which $\leq$, $\geq$, $\top$, and $\bot$ do not appear, then C is 4-satisfiable.*

Proof. Let $\Delta^\mathcal{I} = \{d\}$, $C^\mathcal{I} = \langle \{d\}, \{d\} \rangle$ for each $C \in N_C \cup N_o$, and $a^\mathcal{I} = d$ for each $a \in N_I$. For each $R \in N_R$, let $R^\mathcal{I} = \langle \{(d, d)\}, \{(d, d)\} \rangle$ and $(\exists R.Self)^\mathcal{I} = \langle \{d\}, \{d\} \rangle$. Inducting on the degree of C, if $C \in N_C \cup N_o$ or $C = \exists R.Self$, then the claim holds. Examining the semantics of the concept description operators shows that the claim holds for $\neg C$, $C \sqcup D$, $C \sqcap D$, $\exists R.C$, and $\forall R.C$.

[2] This is so regardless of whether nominals are interpreted classically or as done here.

Proposition 16. *If KB is a $\mathcal{SROIQ}4$ knowledge base in which inequality assertions, $\leq$, and $\geq$, do not appear, then $SF(KB)$ has a 4-model.*

Proof. Using $\mathcal{I}$ above, consider each type of axiom. For $C(a)$ and $R(a,b)$, the claim obviously holds. Since $p^-(R^\mathcal{I}) = \{(d,d)\}$, $\neg R(a,b)$ also holds (we assume that $R \neq U$). It is clear that $Ref(R)$, $Irr(R)$, and $Dis(R,S)$ also hold. Since for all concepts C and D, $C^\mathcal{I} = D^\mathcal{I} = \langle\{d\},\{d\}\rangle$, $C \sqsubset D$ and $C \rightarrow D$ hold. For $C \mapsto D$, $\Delta^\mathcal{I} - p^-(C^\mathcal{I}) = \varnothing$, and so $\mathcal{I}$ 4-models $C \mapsto D$. The cases for RIAs proceed similarly.

One solution to the conflict between cardinality restrictions and nominals is to use *pseudo-nominals*—e.g., $Monday$ instead of $\{monday\}$. In practice, this is sometimes done. However, an ontology obtained via such a substitution is not classically equivalent to the original, and the use of pseudo-nominals sometimes leads to intuitively incorrect results. The same holds in the 4-valued context. Nevertheless, paraconsistency must come at some price, and for many applications the loss of nominals might be an acceptable price to pay.

7 Conclusions, Related Work

This work is a direct extension of earlier research by Ma et al. [9,10,11,12]. The framework described here is closely related to earlier 4-valued description logics developed by Patel-Schneider [16,15], and also Straccia and Meghini [13,14,18]. $\mathcal{SROIQ}4$ is similar to these, but the earlier logics are syntactically closer to $\mathcal{ALC}$ and $\mathcal{SHIQ}$, and to our knowledge nothing similar to the embedding developed by Ma et al. was defined for the older logics. In our opinion, the embedding is the most important contribution. In [12], Ma et al. present $\mathcal{SHIQ}4$ and paraconsistent versions of $\mathcal{EL}^{++}$, Horn-DLs, and DL-Lite logics. An embedding of $\mathcal{SHIQ}4$ into $\mathcal{SHIQ}$ is presented in [12], and it is asserted that it preserves satisfiability. $\mathcal{SROIQ}4$ was first presented in [9] (though lacking role chains, role assertions, and $\exists R.Self$ descriptions; furthermore, roles are treated classically). It's asserted in [9] that embedding $\mathcal{SROIQ}4$ into $\mathcal{SROIQ}$ is consequence preserving, though we believe the proof here is the first full proof given in the literature.[3]

More recently, Zhang, Qi, Ma, and Lin present a paraconsistent variation of $\mathcal{SHIQ}$ based on *quasi-classical logic* [6,19]. It is stated that the logic (QC-$\mathcal{SHIQ}$) remedies problems inherent to the 4-valued approach described here. Specifically, for it, MP, MT, and DS all hold. We point out, however, that while DS fails in $\mathcal{SROIQ}4$ regardless of which inclusion operator used, MP is satisfied by both $\sqsubset$ and $\rightarrow$, and MT is satisfied by $\rightarrow$.

While the failure of DS is indeed a substantial drawback, the primary virtue of the paraconsistent framework extended here is that the embedding into classical logics allows traditional tools to be used. Indeed, we have developed a Java library implementing the translation scheme for $\mathcal{SROIQ}4$. Using the latest version of the OWL API (v3.0) [7], the library allows OWL 2 ontologies–taken as

[3] A full proof for the embedding of $\mathcal{ALC}4$ into $\mathcal{ALC}$ appears in [10].

$\mathcal{SROIQ}4$ knowledge bases—to be transformed according to π. Afterwards, standard OWL reasoners can be used to draw inferences from potentially inconsistent ontologies. Earlier systems in the same vein are ParOWL [10,11], developed for the $\mathcal{ALC}$ fragment of OWL, and a plug-in for the NEON toolkit [17]. To our knowledge, no similar translation scheme or tools exist for $\mathcal{QC}$-$\mathcal{SHIQ}$.

The utility of a paraconsistent reasoner for OWL is perhaps greater than one might initially suspect. While paraconsistent logics are in general weaker than their classical counterparts, we have shown that classical reasoning can be simulated to a point by adding additional axioms to knowledge bases. As this is so, paraconsistency can be enforced in a selective manner. We cite this fact as a response to researchers who are inclined to think paraconsistent logics too weak to be of use.

The incompatibility between cardinality restrictions and nominals, noted above, is perhaps the greatest open problem related to the paraconsistent framework described here. An ideal solution would allow the use of both nominals and cardinality restrictions, and it would also simultaneously ensure both paraconsistency and embeddability into a classical logic. Having three of the four features is possible, of course, but so far having all four has proved to be elusive.

References

1. Arieli, O.: Paraconsistent reasoning and preferential entailments by signed quantified Boolean formulae. ACM Trans. on Comput. Logic, Article 18 (2007)
2. Arieli, O., Avron, A.: Reasoning with logical bilattices. Journal of Logic, Language and Information 5(1), 25–63 (1996)
3. Arieli, O., Avron, A.: The value of the four values. Artificial Intelligence 102(1), 97–141 (1998)
4. Belnap, N.D.: How a computer should think. In: Ryle, G. (ed.) Contemporary Aspects of Philosophy, pp. 30–56. Oriel Press (1977)
5. Belnap, N.D.: A useful four-valued logic. In: Epstein, G., Dunn, J. (eds.) Modern Uses of Multiple Valued Logics, pp. 8–37. D. Reidel, Dordrecht (1977)
6. Besnard, P., Hunter, A.: Quasi-classical logic: Non-trivializable classical reasoning from incosistent information. In: Froidevaux, C., Kohlas, J. (eds.) ECSQARU 1995. LNCS, vol. 946, pp. 44–51. Springer, Heidelberg (1995)
7. Horridge, M., Bechhofer, S.: The OWL API: A Java API for Working with OWL 2 Ontologies. In: Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2009), CEUR Workshop Proceedings, Chantilly, VA, United States, October 23-24, vol. 529. CEUR-WS.org (2009)
8. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press, Menlo Park (2006)
9. Ma, Y., Hitzler, P.: Paraconsistent reasoning for OWL 2. In: Polleres, A. (ed.) RR 2009. LNCS, vol. 5837, pp. 197–211. Springer, Heidelberg (2009)
10. Ma, Y., Hitzler, P., Lin, Z.: Paraconsistent reasoning with OWL – Algorithms and the ParOWL reasoner. Technical Report, no. 1390. AIFB, University of Karlsruhe (2006)
11. Ma, Y., Hitzler, P., Lin, Z.: Algorithms for paraconsistent reasoning with OWL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 399–413. Springer, Heidelberg (2007)

12. Ma, Y., Hitzler, P., Lin, Z.: Paraconsistent reasoning for expressive and tractable description logics. In: Proc. of the 21st Int. Workshop on Description Logics (DL2008), CEUR Workshop Proceedings, vol. 353. CEUR-WS.org. (2008)
13. Meghini, C., Sebastiani, F., Straccia, U., Nazionale, C.: MIRLOG: A logic for multimedia information retrieval. In: Crestani, F., Lalmas, M., van Rijsbergen, C.J. (eds.) Information Retrieval: Uncertainty and Logics. Advanced Models for the Representation and Retrieval of Information, pp. 151–185. Kluwer Academic Publishing, Dordrecht (1998)
14. Meghini, C., Straccia, U.: A relevance terminological logic for information retrieval. In: Proc. of the 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR '96), pp. 197–205. ACM, New York (1996)
15. Patel-Schneider, P.F.: A hybrid, decidable, logic-based knowledge representation system. Computational Intelligence 3(1), 64–77 (1987)
16. Patel-Schneider, P.: A four-valued semantics for terminological logics. Artificial Intelligence 38(3), 319–351 (1989)
17. Ji, Q., Haase, P., Qi, G., Hitzler, P., Stadtmüller, S.: RaDON – Repair and Diagnosis in Ontology Networks. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 863–867. Springer, Heidelberg (2009)
18. Straccia, U.: A sequent calculus for reasoning in four-valued description logics. In: Galmiche, D. (ed.) TABLEAUX 1997. LNCS, vol. 1227, pp. 343–357. Springer, Heidelberg (1997)
19. Ma, Y., Zhang, X., Qi, G., Lin, Z.: Quasi-classical semantics for expressive description logics. In: Proc. of the 22nd Int. Workshop on Description Logics (DL2009), CEUR Workshop Proceedings, vol. 477, CEUR-WS.org. (2009)

Redundancy Elimination on RDF Graphs in the Presence of Rules, Constraints, and Queries*

Reinhard Pichler[1], Axel Polleres[2], Sebastian Skritek[1], and Stefan Woltran[1]

[1] Technische Universität Wien
{pichler,skritek,woltran}@dbai.tuwien.ac.at
[2] DERI, National University of Ireland, Galway
axel.polleres@deri.org

Abstract. Based on practical observations on rule-based inference on RDF data, we study the problem of redundancy elimination on RDF graphs in the presence of rules (in the form of Datalog rules) and constraints (in the form of so-called tuple-generating dependencies), as well as with respect to queries (ranging from conjunctive queries up to more complex ones, particularly covering features of SPARQL, such as union, negation, or filters). To this end, we investigate the influence of several problem parameters (like restrictions on the size of the rules, the constraints, and/or the queries) on the complexity of detecting redundancy. The main result of this paper is a fine-grained complexity analysis of both graph and rule minimisation in various settings.

1 Introduction

The Semantic Web promises to enable computers to gather machine readable meta-data in the form of RDF statements published on the Web and make inferences about these statements by means of accompanying standards such as RDFS and OWL2. While complete OWL2 reasoning is hard – and in many cases even inappropriate for Web data [1] – (incomplete) rule-based inference is becoming quite popular and supported by many RDF Stores and query engines: frameworks like GiaBATA [2], Jena, Sesame, OWLIM,[1] etc. allow for custom inference on top of RDF Stores, supporting different rule-based fragments of RDFS and OWL. Several such fragments have been defined in the literature, such as ρDF [3], DLP [4], OWL$^-$ [5], ter Horst's pD* [6], or SAOR [7], and – more recently – the W3C standardised OWL2RL, a fragment of OWL implementable purely in terms of rule-based inference [8]. All these fragments have in common that they are implementable by simple Datalog-like rules over RDF. As an example, let us take (1) the sub-property rule from RDFS [9, Section 7.3, rule rdfs7],

* Preliminary results have been presented at the Alberto Mendelzon Workshop 2010. R. Pichler, S. Skritek and S. Woltran were supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032. A. Polleres was supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

[1] cf. http://jena.sourceforge.net/, http://openrdf.org/, and http://ontotext.com/owlim/

rules (2)–(5) from OWL2RL [9, Section 4.3, rules prp-inv1,prp-symp,prp-spo2] representing inverse properties, symmetric properties, and property chains:[2]

(1) { `S P O . P` *subPropertyOf* `Q . uri(Q)` } $\Rightarrow$ { `S Q O` }
(2) { `S P O . P` *inverseOf* `Q . uri(O)` $\wedge$ `uri(Q)` } $\Rightarrow$ { `O Q S` }
(3) { `S P O . P` *inverseOf* `Q . blank(O)` $\wedge$ `uri(Q)` } $\Rightarrow$ { `O Q S` }
(4) { `S P O . P` *type SymmetricProperty* `. uri(O)` } $\Rightarrow$ { `O P S` }
(5) { `S P O . P` *type SymmetricProperty* `. blank(O)` } $\Rightarrow$ { `O P S` }
(6) { `S` $\mathtt{P}_0$ $\mathtt{O}_1$`. ...` $\mathtt{O}_n$ $\mathtt{P}_n$ `O. P` *propertyChainAxiom* ($\mathtt{P}_0$ ...$\mathtt{P}_n$) } $\Rightarrow$ { `S P O` }

Let G_D be an RDF graph talking about authors and their publications:

(7) G_D = { `<http://semanticweb.org/wiki/Pat_Hayes>` *made* `<http://www.w3.org/TR/rdf-mt/>.`
(8) `<http://semanticweb.org/wiki/Pat_Hayes>` *name* `"Patrick J. Hayes".`
(9) `<http://www.w3.org/TR/rdf-mt/>` *creator* `"Patrick J. Hayes".`}

Moreover, let graph G_O be part of the ontology defining the terms used in G_D:

(10) G_O = { *name subPropertyOf label.*
(11) *inverseOf type SymmetricProperty.*
(12) *made inverseOf maker.*
(13) *maker inverseOf made.*
(14) *creator propertyChainAxiom* (*maker label*). }

When storing the graph $G = G_D \cup G_O$ in an RDF Store that supports inference over rules (1)–(6), different questions of redundancy arise like if some statements may be deleted since they can be inferred by the rules. In our example, e.g. statement (9) as well as statement (13) may be deleted, since they could be reproduced by inference. Similarly, suppose that we transfer the graph $G = G_D \cup G_O$ to a "weaker" RDF Store that only supports rules (1)–(3). Then the question is if we thus loose any inferences. In fact, the answer is no. Interestingly enough, standard rule sets, such as OWL2RL are even known to be non-minimal [8, Section 4.3].

We thus want to be able to answer the general question about redundancy of both triples and rules. However, it is often important to limit the minimisation of RDF graphs in such a way that certain consistency conditions must be preserved. These consistency conditions can be expressed by means of constraints [10]. We shall restrict ourselves here to constraints in the form of so-called *tuple-generating dependency (tgd) constraints*, which are a generalisation of the familiar foreign-key dependencies in the relational database world. Roughly speaking, a tgd may be viewed as a generalised rule "read" as constraint. So, for instance, if we read rules (4)-(5) as constraints, we could say that graph G alone without rules satisfies these constraints, and likewise the closure of G with respect to rules (1)-(3) does. Tgd constraints can be more general than (Horn) rules in that they also

[2] We disregard full URIs for common RDF terms, i.e., we just write e.g. *inverseOf*, for `<http://www.w3.org/2002/07/owl#inverseOf>`, *name* for `<http://xmlns.com/foaf/0.1/name>`, or *creator* for `<http://purl.org/dc/elements/1.1/>`, etc. Further, $(P_1 \dots P_n)$ in RDF is short for a fresh variable X plus additional triples X *first* P_1 . X_1 *rest* X_2. $\dots X_n$ *first* P_n . X_n *rest nil* . using reserved terms *first*, *rest*, *nil*.

allow otherwise unbound, existential variables in the head, possibly occurring in a larger conjunct. That is, tgds are – rather than rules – constraining queries (in the head) "triggered" by bindings coming from a query in the body; for instance, a constraint

(15) { A *made* D } ⇒ { A *label* N . D *creator* N}

would hold only on graphs where everybody who made something also has a declared label and that label is also used to denote the creator. Note that constraint (15) holds on the closure of G with respect to rule (1) but – as opposed to the constraint reading of (4)-(5) – not on G alone.

Next, we are interested in redundancy with respect to queries. This might be particularly relevant for RDF stores that expose a narrow SPARQL query interface. For instance, suppose that, in our example, we are interested only in completeness with respect to the query "SELECT ?D ?L { ?D *maker* ?M . ?M *label* ?L }" which is the SPARQL way of writing a conjunctive query:

(16) { D *maker* M . M *label* L } $\rightarrow ans(D, L)$

In such setting, both rules (3)–(6) as well as triples (9), (11), (13), and (14) can be dropped. Such redundancy elimination is not unique; for instance, keeping triples (11), (13), and rule (4) we could drop (12), still preserving completeness.

The primary goal of our work is a systematic complexity analysis of both graph and rule minimisation under constraints, as well as with respect to queries. To this end, we investigate the influence of several problem parameters (like restrictions on the size of the rules, constraints, and queries) on the complexity of detecting redundancy. A first important step in this investigation has been recently made by Meier [11]. He studied the following problem: Given a graph G, a set $\mathcal{R}$ of rules and a set $\mathcal{C}$ of tgds, can G be reduced to a proper subgraph $G' \subset G$, such that G' still satisfies $\mathcal{C}$ and the closure of G' under $\mathcal{R}$ coincides with the closure of G under $\mathcal{R}$? For the special case that both the rules in $\mathcal{R}$ and the constraints in $\mathcal{C}$ have bounded size (referred to as *b-boundedness*), this problem was shown to be NP-complete in [11]. In this paper, we want to extend the work initiated in [11] and provide a much more fine-grained analysis of the complexity, e.g., by weakening or strengthening restrictions such as b-boundedness and by considering redundancy elimination that only preserves RDF *entailment* (rather than keeping the closure of the original graph under the original rules unchanged) and additionally considering redundancy with respect to queries.

We shall come up with a collection of complexity results, ranging from tractability to Σ_3^P-completeness. Additionally, we address the orthogonal problems of rule minimisation and the problem of reducing rules or triples without preserving completeness of the entire closure, but only ensuring that the answers to certain queries are preserved.

We shall also discuss further variations of the graph and rule minimisation problem. For instance, the rules and tgds in [11] do not allow variables in predicate positions, which is a severe restriction in the sense that many of the common RDF inferences rules are not covered (e.g., all except rules (4) and (5) above). We will not make this restriction, since it can be dropped without significant change of the complexity results.

Organisation of the paper and summary of results. In Section 2, we recall some basic notions and results. A conclusion and an outlook to future work are given in Section 7. Sections 3–6 contain the main results of the paper, namely:

- *Graph Minimisation.* In Section 3, we provide a comprehensive complexity analysis of the RDF graph minimisation problem, both when full reconstruction of the graph or only RDF entailment is required. We study various settings which result from different restrictions on the rules and/or tgds like restricting their size, considering them as fixed, omitting them, or imposing no restrictions at all. Our complexity results range from tractability to Σ_3^P-completeness.
- *Rule Minimisation.* In Section 4, we consider the problem of minimising the set of rules. We show that the problem of finding redundant rules with respect to a given RDF graph is NP-complete for b-bounded rules and not harder than Δ_2^P for arbitrary rules. Note that rule minimisation is closely related to the field of Datalog equivalence and optimisation. We therefore discuss how the large body of results in this area can be fruitfully applied to the problems studied here.
- *Graph Minimisation w.r.t. Queries.* In Section 5, we study how guaranteeing completeness only w.r.t. a given set of conjunctive queries (CQs) or unions of conjunctive queries (UCQs) influences the complexity for each of the above settings. Considering different restrictions on the size of the queries, hardness never exeeds Σ_3^P, but for some settings raises by two levels in the polynomial hierarchy compared to Section 3. Finally we extend our findings to the problem of rule minimisation. We shall also briefly touch on full SPARQL queries beyond unions of conjunctive queries.
- *Problem Variations.* In Section 6, we analyse the complexity of further problems which are either variations of or strongly related to the graph and rule minimisation problems mentioned above. For instance, rather than asking if an RDF graph contains redundant tuples, we consider the problem whether an RDF graph can be reduced below a certain size. We show that this problem is NP-complete also in those settings where the graph minimisation problem is tractable. We also discuss the effect of allowing blank nodes in predicate positions in the Datalog rules.

Due to lack of space, proofs are only sketched. While for most of the hardness proofs we only describe the idea of the reduction, membership proofs are either also informal or even omitted. All proofs are worked out in detail in [12].

2 Preliminaries

Let U, B, and L denote pairwise disjoint alphabets for *URI references*, *Blank nodes* (or variables) and *Literals*, respectively. We denote unions of these sets simply by concatenating their names.[3] An RDF statement (or *triple*) is a statement of the form $(s, p, o) \in UB \times U \times UBL$, and an RDF *graph* is a set of triples. In this paper, we do not distinguish between variables and blank nodes, but just note that blank nodes/variables appearing in the data are understood to be existentially quantified within the scope of the whole RDF graph they appear in.

[3] In this paper, we use a slightly simplified notion of RDF compared to [9], e.g. not considering typed literals separately.

We write elements from B (U) as alphanumeric strings starting with an upper case letter (lower case letter or number), elements from L as quoted strings, and – inspired by the common Turtle [13] syntax – RDF statements as white-space separated triples and RDF graphs as '.' separated lists of triples in curly braces.

It is convenient to define the notion of *entailment* between two RDF graphs via the interpolation lemma from [9, Section 2] rather than in a model-theoretic way: an RDF graph G_1 *entails* G_2, written $G_1 \models G_2$ if a subgraph of G_1 is an instance of G_2, that is, if there exists a graph *homomorphism*, i.e., a blank node mapping $\mu : B \rightarrow UBL$ such that $\mu(G_2) \subseteq G_1$, where $\mu(G)$ denotes the graph obtained by replacing every variable $\mathtt{B} \in B$ with $\mu(\mathtt{B})$. A homomorphism h' is an *extension* of a homomorphism h if $h'(\mathtt{B}) = h(\mathtt{B})$ for all $\mathtt{B}$ on which h is defined. Given G_1, G_2, deciding whether there exists a homomorphism $G_2 \rightarrow G_1$ (thus also $G_1 \models G_2$) is well known to be NP-complete.

We define a *basic graph pattern* (BGP) as a set of generalised triples (s', p', o') $\in UBL \times UBL \times UBL$, a *filter condition* as a conjunct of the unary predicates $\mathtt{uri}(\cdot)$, $\mathtt{blank}(\cdot)$, $\mathtt{literal}(\cdot)$ (denoting the unary relations U, B, and L, respectively). A *filtered basic graph pattern* (FBGP) is a BGP conjoined with a filter condition, the latter containing only variables already appearing in the BGP. Given an FBGP P, we write $BGP(P)$ and $F(P)$ to denote its components, i.e. its BGP and its filter condition, respectively.

We define an *RDF tuple-generating dependency (tgd) constraint* (or simply constraint) r as $\mathcal{A}nte \Rightarrow \mathcal{C}on$, where the *antecedent* $\mathcal{A}nte$ is an FBGP and the *consequent* $\mathcal{C}on$ is a BGP. A constraint $\mathcal{A}nte \Rightarrow \mathcal{C}on$ is a short-hand notation for the first-order formula $\forall \boldsymbol{x}\big(\mathcal{A}nte(\boldsymbol{x}) \rightarrow (\exists \boldsymbol{y})\mathcal{C}on(\boldsymbol{x}, \boldsymbol{y})\big)$ (where $\boldsymbol{y}$ denotes the blank nodes occurring in $\mathcal{C}on$ only, while $\boldsymbol{x}$ are the remaining blank nodes) Hence, a constraint $\mathcal{A}nte \Rightarrow \mathcal{C}on$ is satisfied over an RDF graph G if for each homomorphism on $\boldsymbol{x}$ mapping BGP($\mathcal{A}nte$) to G, there exists an extension h' of h to $\boldsymbol{y}$ s.t. $h'(\mathcal{C}on) \subseteq G$. To increase the readability, we will sometimes explicitly write out the quantifiers and variable vectors. *RDF rules* (or simply rules), are syntactically restricted constraints, where all variables appearing in $\mathcal{C}on$ also appear in $\mathcal{A}nte$ (akin to the common notion of safety [14] in Datalog). In the following, we will call RDF rules with an empty filter condition *Datalog rules*.[4] We define the closure of a graph G with respect to a set $\mathcal{R}$ of rules, written $Cl_{\mathcal{R}}(G)$ as usual by the least fix-point of the immediate consequence operator. For a given graph G or a given set $\mathcal{R}$ of rules, we use X_G,$X_{\mathcal{R}}$ ($X \in \{U, B, L\}$) to denote the subset of U (resp. B, L) used in G, or $\mathcal{R}$, respectively.

A *conjunctive query (CQ)* over an RDF graph G is of the form $G_q \rightarrow ans(\boldsymbol{X})$, where G_q is an FBGP, *ans* is a distinguished predicate, and $\boldsymbol{X}$ is a vector of blank nodes. We refer to G_q as the *body* of q ($body(q)$), and to $ans(\boldsymbol{X})$ as the *head* of q ($head(q)$). A *union of conjunctive queries (UCQs)* is a set of CQs, all having the same head. The result of a CQ q over some RDF graph G is defined as the set $q(G) = \{(\boldsymbol{x}) \mid \text{for all } x_i \in \boldsymbol{x}\colon x_i \in U_G B_G L_G U_q L_q,\ \text{there exists a homomorphism } \tau\colon B_q \rightarrow U_G B_G L_G \text{ s.t. } \tau(body(q)) \subseteq G \text{ and } \boldsymbol{x} = \tau(\boldsymbol{X})\}$. The result of a UCQ is the union of the results of its CQs.

[4] In fact, we will for most parts of the paper only consider Datalog rules, but will revisit the extension to arbitrary RDF rules in the end of Section 6, concluding that this extension does not change any of our results.

We say that a rule or constraint is *b-bounded* if both antecedent and consequent contain at most b triples. We say a conjunctive query q is *body-b-bounded* if $body(q)$ is b-bounded, and we denote q as *head-b-bounded* if $|\boldsymbol{X}| \leq b$ for some constant b (however, $body(q)$ may be arbitrary). A set $\mathcal{Q}$ of (U)CQs is body-b-bounded (resp. head-b-bounded) if every $q \in \mathcal{Q}$ is body-b-bounded (resp. head b-bounded). Finally, we write $[n]$ to denote the set $\{1, \ldots, n\}$.

3 RDF Graph Minimisation

In this section, we study the complexity of RDF graph minimisation. For different restrictions on the input parameters, the complexity varies between tractability and Σ_3^P-completeness. Formally, we consider the following two basic problems:

Definition 1. *Let* MINI-RDF$^{\models}(G, \mathcal{R}, \mathcal{C})$ *be the following decision problem:*
INPUT: RDF graph G, set $\mathcal{R}$ of RDF rules, set $\mathcal{C}$ of tgds (G satisfies $\mathcal{C}$).
QUESTION: Is there a $G' \subset G$ s.t. $Cl_{\mathcal{R}}(G') \models Cl_{\mathcal{R}}(G)$ and G' satisfies $\mathcal{C}$?

Definition 2. *Let* MINI-RDF$^{\subseteq}(G, \mathcal{R}, \mathcal{C})$ *be the following decision problem [11]:*
INPUT: RDF graph G, set $\mathcal{R}$ of RDF rules, set $\mathcal{C}$ of tgds (G satisfies $\mathcal{C}$).
QUESTION: Is there a $G' \subset G$ s.t. $Cl_{\mathcal{R}}(G) = Cl_{\mathcal{R}}(G')$ and G' satisfies $\mathcal{C}$?

The MINI-RDF$^{\subseteq}$ problem and the minimisation of RDF graphs via entailment aim at two kinds of redundancy elimination: In MINI-RDF$^{\subseteq}$, triples which can be restored via the rules are considered as redundant while minimisation via entailment allows us to replace a graph G by $\bar{G} \subset G$ if $\bar{G} \models G$ holds, i.e. checks if G is lean (see [15]). The MINI-RDF$^{\models}(G, \mathcal{R}, \mathcal{C})$ problem combines these two approaches and thus yields the strongest redundancy criterion. Nevertheless, in most cases, its complexity is not higher than for MINI-RDF$^{\subseteq}$ (see Theorem 1).

Table 1. The complexity of MINI-RDF$^{\models}$ and MINI-RDF$^{\subseteq}$ w.r.t. input parameters ("bb" indicates the set to be b-bounded, and "arb." allows for arbitrary sets.)

		MINI-RDF$^{\models}$	MINI-RDF$^{\subseteq}$
(1)	$\mathcal{R}$ arb., $\mathcal{C}$ arb.	Σ_3^P-complete	Σ_3^P-complete
(2)	$\mathcal{R}$ arb., $\mathcal{C}$ bb	NP-complete	NP-complete
(3)	$\mathcal{R}$ arb., $\mathcal{C}$ fixed	NP-complete	NP-complete
(4)	$\mathcal{R}$ arb., $\mathcal{C} = \emptyset$	NP-complete	NP-complete
(5)	$\mathcal{R}$ bb., $\mathcal{C}$ arb.	Σ_3^P-complete	Σ_3^P-complete
(6)	$\mathcal{R}$ bb, $\mathcal{C}$ bb	NP-complete	NP-complete [11]
(7)	$\mathcal{R}$ bb, $\mathcal{C}$ fixed	NP-complete	NP-complete
(8)	$\mathcal{R}$ bb, $\mathcal{C} = \emptyset$	NP-complete	in P
(9)	$\mathcal{R}$ fixed, $\mathcal{C}$ arb.	Σ_3^P-complete	Σ_3^P-complete
(10)	$\mathcal{R}$ fixed, $\mathcal{C}$ bb	NP-complete	NP-complete
(11)	$\mathcal{R}$ fixed, $\mathcal{C}$ fixed	NP-complete	NP-complete
(12)	$\mathcal{R}$ fixed, $\mathcal{C} = \emptyset$	NP-complete	in P

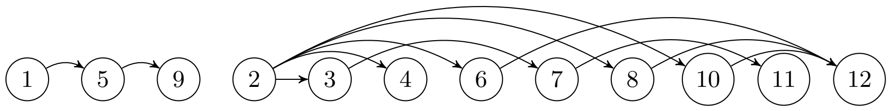


Fig. 1. Dependency graph: Numbers refer to lines in Table 1. An arrow from A to B means that B is a special case of A.

It is easy to see that the condition $Cl_{\mathcal{R}}(G) = Cl_{\mathcal{R}}(G')$ in Definition 2 is equivalent to $G \subseteq Cl_{\mathcal{R}}(G')$. The following lemma shows that similarly, for MINI-RDF$^{\models}$, it is enough to show $Cl_{\mathcal{R}}(G') \models G$ rather than $Cl_{\mathcal{R}}(G') \models Cl_{\mathcal{R}}(G)$.

Lemma 1. *Let G_1, G_2 be RDF graphs and $\mathcal{R}$ a set of rules. Then the following equivalence holds: $Cl_{\mathcal{R}}(G_2) \models Cl_{\mathcal{R}}(G_1) \Leftrightarrow Cl_{\mathcal{R}}(G_2) \models G_1$.*

Theorem 1. *For* MINI-RDF$^{\models}$ *and* MINI-RDF$^{\subseteq}$*, the complexity w.r.t. different assumptions on the input (arbitrary, b-bounded, or fixed rule set; arbitrary, b-bounded, fixed, or no constraints) is as depicted in Table 1.*

The following lemma justifies that we do not have to give an explicit completeness proof for each entry in Table 1, and points out a proof plan for Theorem 1.

Lemma 2. *The graph in Figure 1 correctly describes the dependencies between the problems (identified by their line number) in Table 1, i.e.: If there is an arrow from A to B, then B is a special case of A.*

Hence an arrow from A to B means that membership results for A hold also for B, and that hardness results for B apply also to A. Therefore, to prove Theorem 1, it suffices to show the membership for (1),(2),(8) and the hardness for (4),(9),(11),(12). Due to lack of space, we only work out the hardness results for (9) and (11) (the latter only for MINI-RDF$^{\subseteq}$). Before, we shortly discuss the membership results and give an intuition of why they are correct. All proofs are worked out in detail in the full paper [12].

The most general case, (1), can be solved by a guess and check algorithm that is allowed to use a Π_2^P oracle for the checks. One has to guess: a subgraph G' of G, a sequence of rule applications on G', and for each rule application a homomorphism justifying that the rule is applicable. Note that $Cl_{\mathcal{R}}(G') \subseteq AD^3$ (with $AD = U_G U_{\mathcal{R}} B_G B_{\mathcal{R}} L_G L_{\mathcal{R}}$). Hence if considering all possible rule applications of length $|AD|^3$, one of them has to return $Cl_{\mathcal{R}}(G')$. The most expensive check is to test if G' satisfies $\mathcal{C}$. However, it obviously fits into Π_2^P.

The following properties lead to the cases of lower complexity: If $\mathcal{R}$ is a b-bounded set, then $Cl_{\mathcal{R}}(G')$ can be computed in polynomial time [11, Proposition 9] and if $\mathcal{C}$ is a b-bounded set, then testing if G' satisfies $\mathcal{C}$ is in PTIME [11, Proposition 3]. For the tractable cases, note that if $\mathcal{C} = \emptyset$, then not all subgraphs of G have to be checked, but only those missing exactly one triple from G.

Lemma 3. *The problems* MINI-RDF$^{\models}(G, \mathcal{R}, \mathcal{C})$ *and* MINI-RDF$^{\subseteq}(G, \mathcal{R}, \mathcal{C})$*, for fixed $\mathcal{R}$ and arbitrary $\mathcal{C}$, are Σ_3^P-hard.*

Proof. Σ_3^P*-hardness* is shown by reduction from the well-known Σ_3^P-complete problem QSAT$_3$, of which we only give an informal description here. Let an instance of QSAT$_3$ be given by $F = \exists \boldsymbol{x_1} \forall \boldsymbol{y_1} \exists \boldsymbol{x_2} \bigwedge_{i=1}^n C_i$, with $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ (clearly, the restriction to 3-CNF is w.l.o.g.). The graph G created contains on the one hand triples encoding truth assignments on clauses (e.g. $\{0\ h_1\ a_{001} \,.\, 0\ h_2\ a_{001}\ \,.\, 1\ h_3\ a_{001}\}$ for the assignment (*false*, *false*, *true*)), and on the other hand triples encoding the two possible truth assignments for variables (e.g. $\{v_i\ q_1\ a_{01} \,.\, v_i\ q_1\ a_{10}\}$ for $x_i \in \boldsymbol{x_1}$ where v_i is a new URI for each x_i and the URI a_{01} (resp. a_{10}) denotes that x_i evaluates to *false*, hence $\neg x_i$ evaluates to *true* (resp. x_i to *true* and $\neg x_i$ to *false*), together with further triples that allow us to actually refer to the truth value of x_i (resp. $\neg x_i$)) under a selected truth assignment. The rules and constraints are chosen in such a way that (1) the triples encoding the truth assignment (*false*, *false*, *false*) for clauses must not be present in any valid subgraph $G' \subset G$, (2) for every $x_i \in \boldsymbol{x_1}$ exactly one of the two triples encoding a truth assignment must be present in G' and (3) for all other variables, both triples have to remain in G'. The restrictions imposed by $\bigwedge_{i=1}^n C_i$ are encoded in one big tgd, where every homomorphism from its antecedent to G' defines a truth assignment for $\boldsymbol{x_1}$ and $\boldsymbol{y_1}$. Thereby for every valid G' all such homomorphisms define the same truth assignment on $\boldsymbol{x_1}$, hence the values for $\boldsymbol{x_1}$ are determined by the selection of G'. But every homomorphism defines a different truth assignment on $\boldsymbol{y_1}$, and there exists exactly one homomorphism for each of the $2^{|\boldsymbol{y_1}|}$ truth assignments on $\boldsymbol{y_1}$. The consequent of the tgd contains a representation of the literals in each clause C_i and has the following property: for every homomorphism h from the antecedent to G', there exists an extension of h to a homomorphism h' from the consequent to G' iff this extension defines a truth assignment on $\boldsymbol{x_2}$ such that the assignment on $\boldsymbol{x_1}$, $\boldsymbol{y_1}$ and $\boldsymbol{x_2}$ maps the representations of the clauses onto the possible truth assignments for clauses present in G'. As all triples encoding these truth assignments must be in G', except the ones for (*false*, *false*, *false*) which must not, such an extension for every homomorphism from the antecedent to G' implies that F is valid. □

Lemma 4. *The problems* MINI-RDF$^{\subseteq}(G, \mathcal{R}, \mathcal{C})$ *and* MINI-RDF$^{\models}(G, \mathcal{R}, \mathcal{C})$*, where both* $\mathcal{R}$ *and* $\mathcal{C}$ *are considered to be fixed, are* NP*-hard.*

Proof. As NP-hardness of MINI-RDF$^{\models}$ follows easily from the co-NP-hardness of testing if G is lean [15], we concentrate on MINI-RDF$^{\subseteq}$ and prove its NP-hardness by reduction from the 3-SAT problem. We fix the rules and tgds as

$$\begin{aligned}
\mathcal{R} &= \big\{\ \{X'\ \mathit{in}\ I \,.\, X\ \mathit{active}\ I\} \Rightarrow \{X'\ \mathit{active}\ I\}\big\}\\
\mathcal{C} &= \big\{\ \{X\ \mathit{active}\ I \,.\, X\ \mathit{in}\ J\} \Rightarrow \{X\ \mathit{active}\ J\}\\
&\qquad \{X\ \mathit{clash}\ X' \,.\, X\ \mathit{active}\ I \,.\, X'\ \mathit{active}\ I' \,.\, Y\ \mathit{in}\ J\} \Rightarrow \{Y\ \mathit{active}\ J\}\big\}.
\end{aligned}$$

Now let an instance of 3-SAT be given by the formula $F = C_1 \wedge \cdots \wedge C_n$, where $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ and the $l_{i,j}$ are literals. W.l.o.g., we assume that every variable appears negated and unnegated in F. Then we construct an RDF graph $G = \{l^*_{i,j}\ \mathit{in}\ c_i \mid i \in [n], j \in [3]\} \cup \{l^*_{i,j}\ \mathit{active}\ c_i \mid i \in [n], j \in [3]\} \cup \{x_j\ \mathit{clash}\ \bar{x}_j \mid x_j \text{ in } F\}$, where we introduce new URIs c_i (for every clause C_i) and $x_j, \bar{x}_j$ (for every variable x_j in F), and $l^*_{i,j} = x_j$ (resp. $\bar{x}_j$) if $l_{i,j} = x_j$ (resp. $\neg x_j$).

Intuitively, the triples in G with predicate *in* encode the literals in F. If a triple with predicate *active* remains in the selected subgraph G' then the corresponding literal in F is set to true. The triples with *clash* keep track of dual literals. $\square$

4 Rule Minimisation

In this section, we study the rule minimisation problem of RDF graphs. Although there is a huge amount of literature in the Datalog world addressing related problems (as query containment), the particular nature of the problems we study requires a distinguished complexity analysis. Note that rules for RDF, when written as Datalog rules, have a fixed predicate arity of three, which makes problems computationally easier than in the general Datalog setting (see, e.g. [16]). Depending on whether we consider the Datalog rules as b-bounded or not, we obtain complexity results from NP-completeness to Δ_2^P-membership. The rule minimisation problem is formally defined as follows. As the RDF graph remains unchanged, constraints are irrelevant here.

Definition 3. *Let* RDF-RULEMIN$^{\models}(G, \mathcal{R})$ *be the following decision problem:*
INPUT: An RDF graph G and a set $\mathcal{R}$ of RDF rules.
QUESTION: Does there exist $\mathcal{R}' \subset \mathcal{R}$ s.t. $Cl_{\mathcal{R}'}(G) \models Cl_{\mathcal{R}}(G)$?

Definition 4. *Let* RDF-RULEMIN$^{\subseteq}(G, \mathcal{R})$ *be the following decision problem:*
INPUT: An RDF graph G and a set $\mathcal{R}$ of RDF rules.
QUESTION: Does there exist $\mathcal{R}' \subset \mathcal{R}$ s.t. $Cl_{\mathcal{R}'}(G) = Cl_{\mathcal{R}}(G)$?

For the case that the set of rules is b-bounded, we can pinpoint the complexity of the problem to NP.

Theorem 2. *For a set $\mathcal{R}$ of b-bounded rules (for fixed b), the problem* RDF-RULEMIN$^{\models}(G, \mathcal{R})$ *is* NP*-complete while* RDF-RULEMIN$^{\subseteq}(G, \mathcal{R})$ *is in* PTIME*.*

Proof. The hardness is shown by reduction from the 3-Colorability problem. The RDF graph G is built over the URIs $U = \{0, 1, 2\}$ in subject and object positions. G contains triples of the form $i\ e\ j$ for all value combinations $i, j \in U$ with $i \neq j$. $\mathcal{R}$ contains a single rule which generates an encoding $X_\alpha\ e\ X_\beta$ (with blank nodes X_α, X_β) for each edge (v_α, v_β) of the graph to be 3-colored. This rule is redundant iff a valid 3-coloring exists, i.e., iff the triples $X_\alpha\ e\ X_\beta$ can be mapped into $\{i\ e\ j \mid i \neq j\}$.

For the membership, note that it suffices to compare the closure of G under $\mathcal{R}$ with the closure of G under every subset of $\mathcal{R}$ missing exactly one rule. In the b-bounded case, the closure can be computed efficiently. Hence, we get PTIME-membership for RDF-RULEMIN$^{\subseteq}$ and NP-membership for RDF-RULEMIN$^{\models}$ (the NP-computation is needed only for the entailment check). $\square$

Theorem 3. *For arbitrary rules,* RDF-RULEMIN$^{\subseteq}(G, \mathcal{R})$ *is* co-NP*-hard and in Δ_2^P while* RDF-RULEMIN$^{\models}(G, \mathcal{R})$ *is* NP*-hard,* co-NP*-hard, and in Δ_2^P.*

Proof. The Δ_2^P upper bound is due to the fact that computing the closure under a set of arbitrary rules requires an NP-oracle (to check if a rule is applicable). The NP-hardness of RDF-RULEMIN$^{\models}(G, \mathcal{R})$ carries over from Theorem 2. The co-NP-hardness of both problems is shown by a straightforward reduction from the co-problem of 3-Colorability: $\mathcal{R}$ contains a single rule whose body encodes the graph to be 3-colored. This rule is redundant iff no 3-coloring exists. $\square$

In order to reduce the complexity of the problems RDF-RULEMIN$^{\subseteq}(G, \mathcal{R})$ and RDF-RULEMIN$^{\models}(G, \mathcal{R})$, one could seek for approximations of those problems. In fact, one option is to check for *redundant* rules in the set $\mathcal{R}$ of given Datalog rules; or whether some rule is subsumed by another rule from $\mathcal{R}$. The first problem is known to be tractable while the test for rule subsumption is NP-complete (see [17]). The latter result can be shown to hold also for rules of bounded arity (which we deal with here); but becomes tractable in the case of b-bounded rules. Further methods (e.g., folding and unfolding of rules) are well understood for logic programs (see [18]), and could also apply to our domain. An in-depth analysis how to use those results in our setting is left for future work.

5 Minimisation w.r.t. Queries

Another variant of the RDF graph and rule minimisation problems is to guarantee completeness only w.r.t. a given set of queries. We restrict ourselves here to (unions of) conjunctive queries (CQs resp. UCQs). Such a minimisation is of high interest, e.g. when importing data into an RDF Store that provides a narrow query interface only. Formally, we get the following problems:

Definition 5. MINI-RDF$^{\subseteq,CQ}(G, \mathcal{R}, \mathcal{C}, \mathcal{Q})$ *is the following decision problem:*
INPUT: An RDF graph G, a set $\mathcal{R}$ of RDF rules, a set $\mathcal{C}$ of tgds (G satisfies $\mathcal{C}$), and a set $\mathcal{Q}$ of CQs.
QUESTION: Is there a $G' \subset G$ s.t. (1) for every $q \in \mathcal{Q}$, the answers to q over $Cl_{\mathcal{R}}(G)$ coincide with the answers to q over $Cl_{\mathcal{R}}(G')$ and (2) G' satisfies $\mathcal{C}$?

Definition 6. RDF-RULEMIN$^{\subseteq,CQ}(G, \mathcal{R}, \mathcal{Q})$ *is the following decision problem:*
INPUT: An RDF graph G, a set $\mathcal{R}$ of RDF rules, and a set $\mathcal{Q}$ of CQs.
QUESTION: Is there a $\mathcal{R}' \subset \mathcal{R}$ s.t. for every $q \in \mathcal{Q}$, the answers to q over $Cl_{\mathcal{R}}(G)$ coincide with the answers to q over $Cl_{\mathcal{R}'}(G)$?

Note that, in the above problem definitions, $\mathcal{Q}$ is *some* set of CQs. If we choose $\mathcal{Q}$ to be the set of *all* CQs, then MINI-RDF$^{\subseteq,CQ}$ coincides with MINI-RDF$^{\subseteq}$ and MINI-RDF$^{\subseteq,CQ}$ coincides with RDF-RULEMIN$^{\subseteq}$. Actually, this is the case for any set $\mathcal{Q}$ containing the CQ $\{S\ P\ O\} \rightarrow ans(S, P, O)$. It follows immediately that all hardness results from Sections 3 and 4 carry over to the CQ-variants.

Analogously to the settings studied in the previous sections resulting from different restrictions on $\mathcal{C}$ and $\mathcal{R}$, we also study three settings of the CQ-variants of these problems by considering $\mathcal{Q}$ to be body-b-bounded, head-b-bounded, or unrestricted, respectively. We thus get the following complexity results.

Theorem 4. *For* MINI-RDF$^{\subseteq,CQ}$*, the complexity w.r.t. different assumptions on the input (arbitrary, b-bounded or fixed rule set; arbitrary, b-bounded, fixed, or*

Table 2. The complexity of MINI-RDF$^{\subseteq,CQ}$ (1-12) and RDF-RULEMIN$^{\subseteq,CQ}$ (I. - II.) w.r.t. input parameters ("bb" stands for "b-bounded", and "arb." for "arbitrary")

		$\mathcal{Q}$ body-bb (a)	$\mathcal{Q}$ head-bb (b)	$\mathcal{Q}$ arb. (c)
(1)	$\mathcal{R}$ arb., $\mathcal{C}$ arb.	Σ_3^P-complete	Σ_3^P-complete	Σ_3^P-complete
(2)	$\mathcal{R}$ arb., $\mathcal{C}$ bb	NP/ Δ_2^P	NP/ Δ_2^P	Σ_3^P-complete
(3)	$\mathcal{R}$ arb., $\mathcal{C}$ fixed	NP/ Δ_2^P	NP/ Δ_2^P	Σ_3^P-complete
(4)	$\mathcal{R}$ arb., $\mathcal{C} = \emptyset$	NP/ Δ_2^P	NP/ Δ_2^P	Π_2^P-complete
(5)	$\mathcal{R}$ bb., $\mathcal{C}$ arb.	Σ_3^P-complete	Σ_3^P-complete	Σ_3^P-complete
(6)	$\mathcal{R}$ bb, $\mathcal{C}$ bb	NP-complete	NP/ Δ_2^P	Σ_3^P-complete
(7)	$\mathcal{R}$ bb, $\mathcal{C}$ fixed	NP-complete	NP/ Δ_2^P	Σ_3^P-complete
(8)	$\mathcal{R}$ bb, $\mathcal{C} = \emptyset$	in P	NP/ Δ_2^P	Π_2^P-complete
(9)	$\mathcal{R}$ fixed, $\mathcal{C}$ arb.	Σ_3^P-complete	Σ_3^P-complete	Σ_3^P-complete
(10)	$\mathcal{R}$ fixed, $\mathcal{C}$ bb	NP-complete	NP/ Δ_2^P	Σ_3^P-complete
(11)	$\mathcal{R}$ fixed, $\mathcal{C}$ fixed	NP-complete	NP/ Δ_2^P	Σ_3^P-complete
(12)	$\mathcal{R}$ fixed, $\mathcal{C} = \emptyset$	in P	NP/ Δ_2^P	Π_2^P-complete
(I.)	$\mathcal{R}$ arb.	co-NP/ Δ_2^P	co-NP+ NP/ Δ_2^P	Π_2^P-complete
(II.)	$\mathcal{R}$ bb.	in P	NP/ Δ_2^P	Π_2^P-complete

no constraints; body-b-bounded, head b-bounded, or arbitrary CQs) is as depicted in Table 2, rows (1) – (12). Likewise, the complexity of RDF-RULEMIN$^{\subseteq,CQ}$ *is depicted in Table 2, rows (I) – (II).*

Thereby (co-)NP / Δ_2^P *denotes the lower bound / upper bound for the complexity. We write* co-NP+ NP/ Δ_2^P *if both,* co-NP- *and* NP- *hardness hold. All lower bounds hold even if Q consists of a single CQ. Likewise, all upper bounds hold even if $\mathcal{Q}$ is a set of UCQs.*

Obviously, body-b-bounded (U)CQs are a special case of head-b-bounded (U)CQs, which in turn are a special case of arbitrary (U)CQs. By combining this observation with Lemma 2, to prove Theorem 4, it suffices to show membership for the entries (6a), (8a), (2b), (1c), (4c) as well as (Ia), (IIb), and (IIc) in Table 2, and hardness for (11a), (12b), (11c), (12c) as well as (IIa), (Ib), and (Ic). Due to space restrictions, we only give a rough sketch of the intuition of these results. All proofs are worked out in detail in the full version [12].

Membership of the most general case (1c) is shown by considering the following algorithm: guess a subset $G' \subset G$ and check with Π_2^P-oracles if G' satisfies $\mathcal{C}$ and if $q(\hat{G}) = q(\hat{G}')$, where $\hat{G} = Cl_{\mathcal{R}}(G)$, resp. $\hat{G}' = Cl_{\mathcal{R}}(G')$. Moreover, the closures $\hat{G}$ and $\hat{G}'$ can be computed in Δ_2^P, since they are subsets of AD^3.

The other columns contain potentially easier settings because of the restrictions on the queries, while the other rows are potentially easier because of restrictions on $\mathcal{R}$ and $\mathcal{C}$. In particular, if no constraints are present, it suffices to check the "direct" subsets $G' = G \setminus \{t\}$ for each $t \in G$. Thus the non-deterministic guess of $G' \subset G$ is no longer needed. By the same token, rule minimisation is not harder than Π_2^P, since we only need to check the direct subsets $\mathcal{R}' = \mathcal{R} \setminus \{r\}$. If the queries are head-b-bounded, then there are at most polynomially many candidates for answer-tuples. Hence, to answer a query q over two different RDF graphs is feasible in Δ_2^P (rather than Π_2^P). For body-b-bounded queries, the answers to a query q over an RDF graph can even be computed in PTIME.

Turning to the lower bounds, the NP-hardness for (11a) follows immediately from the above remark that the hardness results of MINI-RDF$^{\subseteq}$ carry over. (12b) differs from the previous setting by allowing more expressive queries, but no constraints (which, for MINI-RDF$^{\subseteq}$, leads to tractability). However, the NP-hardness of this case follows immediately from the co-NP-hardness of checking if an RDF graph is lean [15] and defining $\mathcal{Q} = \{G \rightarrow \mathit{ans}()\}$. The hardness for (11c) is shown by reduction from QSAT$_3$. Its main idea is, given a formula $F = \exists \boldsymbol{x}_1 \forall \boldsymbol{y}_1 \exists \boldsymbol{x}_2 \phi$, to define a CQ q and a graph G such that every homomorphism $\tau \colon \mathit{body}(q) \rightarrow G$ defines a truth assignment on the variables in F. (The proof allows even $\mathcal{R} = \emptyset$.) Thereby q outputs the values of this truth assignment on $\boldsymbol{y}_1$. G is further chosen in such a way that $q(G)$ contains an encoding of all possible truth assignments on $\boldsymbol{y}_1$. The constraints in $\mathcal{C}$ are such that over every proper subgraph $G' \subset G$ that satisfies $\mathcal{C}$, every homomorphism from $\mathit{body}(q)$ to G' now encodes truth assignments that actually satisfy ϕ. At the same time, the assignment on $\boldsymbol{x}_1$ is already defined by the choice of G'. Hence, if $q(G')$ also contains encodings for all possible truth assignments on $\boldsymbol{y}_1$, this means that F is indeed satisfied. For $\mathcal{C} = \emptyset$, we only get Π_2^P-hardness since we can no longer express that valid choices for G' encode a truth assignment on $\boldsymbol{x}_1$.

For the rule minimisation, the Π_2^P-hardness is shown similarly to the Π_2^P-hardness in case (12c). In case of (head-/body-)b-bounded queries, the answers to the queries can no longer produce all possible truth assignments on $\boldsymbol{y}_1$. Hence, we can only prove NP- and co-NP-hardness, respectively, in cases (Ia) and (IIa).

5.1 Beyond Conjunctive Queries – SPARQL

RDF minimisation w.r.t. (unions of) conjunctive queries could be extended to more expressive query languages. Actually, it can be checked that all upper bounds proved in this Section are still valid if the CQs are allowed to contain negation in the body. In particular, the complexity of the problems considered here does not go beyond Σ_3^P for this kind of extension, cf. [12]. In contrast, if we allow arbitrary non-recursive datalog queries with negation (a query language which – as well known – covers all of SPARQL [19]), then the complexity of the problems considered here will be dominated by the complexity of query evaluation, which is PSPACE-complete in this case, see [20]. We leave a more fine-grained analysis of different fragments of SPARQL to future work.

6 Problem Variations

In this section, we discuss some further problems which are variations of or strongly related to the problems studied in the previous sections. We start by a variation of the graph minimisation problem. But now we ask if G can be replaced by a subgraph G' whose size is bounded by some given bound k (rather than an arbitrary subgraph $G' \subset G$). Formally, we study the following problem.

Definition 7. *Let* MINI-RDF$^{card}(G, \mathcal{R}, \mathcal{C}, k)$ *be the following decision problem: INPUT: An RDF graph G, a set $\mathcal{R}$ of RDF rules, a set $\mathcal{C}$ of tgds and integer k. QUESTION: Does there exist a subgraph $G' \subset G$ with $|G'| \leq k$, s.t. G' satisfies $\mathcal{C}$ and $G \subseteq Cl_{\mathcal{R}}(G')$?*

It can be easily verified that for all cases in Table 1 that are at least NP-hard, the complexity for MINI-RDF^{card} does not change. Intuitively, this is because the nondeterministic algorithms for solving these problems all start with "guess a subgraph $G' \subset G$", which can be easily changed to "guess a subgraph with at most k triples". Therefore, the only two interesting cases are MINI-RDF$^{\subseteq}$ with a b-bounded or fixed set $\mathcal{R}$ and no constraints, as they can be decided in PTIME. We show that for MINI-RDF^{card}, the complexity goes up to NP-completeness.

Theorem 5. *The problem* MINI-RDF$^{card}(G, \mathcal{R}, \mathcal{C}, k)$ *is* NP*-complete if $\mathcal{C} = \emptyset$ and $\mathcal{R}$ is either considered as fixed or a set of b-bounded rules (for fixed b).*

Proof. The *hardness* proof is by reduction from the Vertex Cover problem. We give the basic ideas of this reduction. Given some graph $G = (V, E)$, the RDF graph G^{rdf} contains one distinct triple for every $v \in V$. The intuition is that the subset of those triples contained in a valid subgraph $G' \subset G^{rdf}$ describes a vertex cover. We further have three rules, one that (given $G' \subset G$) adds all edges covered by the remaining vertices in G', one that (by repeated application) checks whether all edges are covered, and finally one rule that, if indeed all edges are covered, allows to restore the vertices from $G^{rdf} \setminus G'$. To allow to express according rules, G^{rdf} contains triples encoding further information (like e.g. neighbourhood of vertices and edges). But as they cannot be derived by any rule, they must remain unchanged in any valid $G' \subset G^{rdf}$. Further, their number (say K) only depends on G, such that there exists a vertex cover of size k iff there exists a valid $G' \subset G^{rdf}$ of size $K + k$. □

Next we want to identify the sources of the complexity of MINI-RDF$^{\models}$ and MINI-RDF$^{\subseteq}$ for the cases where $\mathcal{C}$ is allowed to contain arbitrary tgds. We show that the complexity is independent of the rules, but arises mainly from the question whether there exists some non-empty subgraph that satisfies all constraints.

Theorem 6. *Let G be a RDF graph and $\mathcal{C}$ a set of tgds. Deciding whether there exists some $\emptyset \neq G' \subset G$ s.t. G' satisfies $\mathcal{C}$ is Σ_3^P-complete.*

Proof. Membership follows from Theorem 1. Hardness is shown by a modification of the reduction given in the proof of Lemma 3. We give the intuition of these modifications. In the aforementioned proof, the intuitive meaning of the rules, together with the requirement $G \subseteq Cl_{\mathcal{R}}(G')$, was that for each $v_i \in \boldsymbol{x_1}$, either $\{v_i\ q_1\ a_{01}\}$ or $\{v_i\ q_1\ a_{10}\}$ has to remain in the subgraph G'. However, this can be also formulated as a constraint. By introducing an additional triple for every $v_i \in \boldsymbol{x_1}$ (e.g. $\{v_i\ opt\ v_i\}$) that is enforced to be contained in any non-empty subgraph, the tgd $\{V\ opt\ V\} \Rightarrow \{V\ q_1\ A\}$ does the job. □

From the (full) proof of Lemma 4, it follows that for MINI-RDF$^{\models}$, one source of the NP-hardness is just to decide the entailment. However, similarly to the last theorem, we can show that for b-bounded tgds, just testing for the existence of a valid subgraph already contains the full hardness too.

Theorem 7. *Let G be an RDF graph and $\mathcal{C}$ a set of b-bounded tgds. Deciding whether there exists some $\emptyset \neq G' \subset G$ s.t. G' satisfies $\mathcal{C}$ is* NP*-complete.*

Proof. Membership follows from Theorem 1. *Hardness* is shown by reduction from the SAT problem. The reduction is very similar to the one of Lemma 4, only that all the implicit information about which triples must not be removed from G (expressed by not providing rules to derive them) now have to be made explicit as tgds. This however no longer allows for a fixed set of tgds, but makes the number of tgds dependent on F. □

Recall that tgds generalise (safe) datalog rules by allowing existential quantification and conjunctions in the head. In other words, datalog rules are an important special case of tgds – referred to as *full tgds* in the information integration literature. Below, we show that restricting the constraints to full tgds pushes the Σ_3^P-completeness results from Theorems 1 and 6 down to Σ_2^P.

Theorem 8. *The problems* MINI-RDF$^{\models}(G, \mathcal{R}, \mathcal{C})$ *and* MINI-RDF$^{\subseteq}(G, \mathcal{R}, \mathcal{C})$ *are Σ_2^P-complete if $\mathcal{C}$ is a set of full tgds. Σ_2^P-completeness even holds for fixed $\mathcal{R}$.*

Likewise, let G be an RDF graph and $\mathcal{C}$ a set of full tgds. Deciding whether there exists some $\emptyset \neq G' \subset G$ s.t. G' satisfies $\mathcal{C}$ is Σ_2^P-complete.

Proof. The Σ_2^P-*membership* is established by the same algorithm as the Σ_3^P-membership in case of unrestricted tgds according to Theorem 1. However, by the restriction to full tgds, we now only need a co-NP-oracle (rather than Π_2^P) for checking that the tgds are satisfied. The Σ_2^P-*hardness* is shown via reduction from QSAT$_2$ by using similar ideas as in the Σ_3^P-hardness proof in Lemma 3. □

So far, we have not commented on the impact of allowing general RDF rules as defined in Section 2, i.e., rules containing additional predicates *uri*(.), *blank*(.), *lit*(.) in the bodies. In the full version of this paper [12], we give a very simple argument that a polynomial time preprocessing suffices to support these predicates naturally in RDF. The same argument allows us to overcome the problem that the closure w.r.t. a rule set $\mathcal{R}$ may contain invalid RDF triples (e.g. due to a blank node in a predicate position). This result holds independently of whether intermediate results are allowed to contain invalid triples or not.

7 Conclusion

We proved a collection of complexity results for minimisation problems over RDF graphs where we considered various restrictions on the rules and tgds. One such restriction was b-boundedness [11]. We note that this restriction can be relaxed by bounding not necessarily the size of the rules (or tgds) but only the maximal number of blank nodes occurring in the rules (or tgds) — in the Datalog world, Vardi [21] showed that such a restriction decreases complexity. We further discussed how the complexity of the problem increases if one requires completeness only with respect to a given set of conjunctive queries (CQs). Notably, if the CQs are restricted to have bounded head arity, while providing additional minimisation potential, the problem becomes only mildly harder.

The minimisation problems considered here are driven by practical needs to represent RDF data compactly or tailor them to engines supporting different rule sets. Our results also provide a basis for eliminating redundancies in existing practically relevant rule sets, such as OWL2RL [8]. We believe that our results will gain even more relevance with the advent of novel standards such as the W3C rule interchange format (RIF) which will allow one to enrich RDFS and OWL with Web-publishable custom rule sets [22].

As future work, our investigations should be further extended in several directions such as a more fine-grained analysis of SPARQL fragments when redundancy w.r.t. queries is considered, for instance well-designed SPARQL queries [20]. Moreover, we plan to cast the obtained results into practical algorithms to "compress" RDF graphs and rule sets, investigate related relevant problems such as "trading" triples for rules, or vice versa, and experimentally evaluating effects of such transformations on query answering with dynamic inference such as sketched in [2].

Finally, the high complexities identified in this paper call for a systematic search for fragments with lower complexity. One step in this direction has already been the restriction to b-boundedness studied in this paper. It is motivated by the assumption that rules, constraints, and queries are usually significantly smaller than the size of the RDF data. Further restrictions (like restrictions on graph parameters like treewidth) and their effect on the complexity of our minimisation problems are left for future work.

References

1. Hogan, A., Decker, S.: On the ostensibly silent 'W' in OWL 2 RL. In: Polleres, A. (ed.) RR 2009. LNCS, vol. 5837, pp. 118–134. Springer, Heidelberg (2009)
2. Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: Dynamic querying of mass-storage RDF data with rule-based entailment regimes. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 310–327. Springer, Heidelberg (2009)
3. Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal deductive systems for RDF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 53–67. Springer, Heidelberg (2007)
4. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logics. In: Proc. WWW'03, pp. 48–57 (2003)
5. de Bruijn, J., Polleres, A., Lara, R., Fensel, D.: OWL^{-}. WSML D20.1v0.2 (2005)
6. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. Web Sem. 3(2-3), 79–115 (2005)
7. Hogan, A., Harth, A., Polleres, A.: Scalable authoritative OWL reasoning for the web. International Journal on Semantic Web and Information Systems 5(2) (2009)
8. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web ontology language profiles. W3C Recommendation (October 2009)
9. Hayes, P.: RDF semantics. Technical report, W3C. W3C Recommendation (February 2004)
10. Lausen, G., Meier, M., Schmidt, M.: SPARQLing constraints for RDF. In: Proc. EDBT'08. ACM Press, New York (2008)

11. Meier, M.: Towards Rule-Based Minimization of RDF Graphs under Constraints. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 89–103. Springer, Heidelberg (2008)
12. Pichler, R., Polleres, A., Skritek, S., Woltran, S.: Minimizing RDF graphs under rules and constraints revisited. Technical report, DERI (April 2010), http://www.deri.ie/fileadmin/documents/DERI-TR-2010-04-23.pdf
13. Beckett, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language, W3C Team Submission (January 2008), http://www.w3.org/TeamSubmission/turtle/
14. Ullman, J.D.: Principles of Database and Knowledge Base Systems. Computer Science Press, New York (1989)
15. Gutierrez, C., Hurtado, C., Mendelzon, A.: Foundations of semantic web databases. In: Proc. PODS'04, pp. 95–106. ACM, New York (2004)
16. Eiter, T., Faber, W., Fink, M., Woltran, S.: Complexity results for answer set programming with bounded predicate arities and implications. Ann. Math. Artif. Intell. 51(2-4), 123–165 (2007)
17. Eiter, T., Fink, M., Tompits, H., Traxler, P., Woltran, S.: Replacements in non-ground answer-set programming. In: Proc. KR'06, pp. 340–351. AAAI Press, Menlo Park (2006)
18. Pettorossi, A., Proietti, M.: Transformation of logic programs. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 5, pp. 697–787. Oxford University Press, Oxford (1998)
19. Angles, R., Gutierrez, C.: The expressive power of SPARQL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 114–129. Springer, Heidelberg (2008)
20. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3) (2009)
21. Vardi, M.: On the complexity of bounded-variable queries. In: Proc. PODS'95, pp. 266–276. ACM Press, New York (1995)
22. de Bruijn, J.: RIF RDF and OWL Compatibility. W3C Proposed Recommendation (May 2010), http://www.w3.org/TR/2010/PR-rif-rdf-owl-20100511/

Defeasibility in Answer Set Programs via Argumentation Theories*

Hui Wan[1], Michael Kifer[1], and Benjamin Grosof[2]

[1] State University of New York at Stony Brook, USA
[2] Vulcan, Inc., USA

Abstract. Defeasible reasoning has been studied extensively in the last two decades and many different and dissimilar approaches are currently on the table. This multitude of ideas has made the field hard to navigate and the different techniques hard to compare. Our earlier work on Logic Programming with Defaults and Argumentation Theories (LPDA) introduced a degree of unification into the approaches that rely on the well-founded semantics. The present work takes this idea further and introduces ASPDA—a unifying framework for defeasibility of *disjunctive* logic programs under the Answer Set Programming (ASP). Since the well-founded and the answer set semantics underlie almost all existing approaches to defeasible reasoning in Logic Programming, LPDA and ASPDA together capture most of those approaches. In addition to ASPDA, we obtained a number of interesting and non-trivial results. First, we show that ASPDA is reducible to ordinary ASP programs, albeit at the cost of exponential blowup in the number of rules. Second, we study reducibility of ASPDA to the non-disjunctive case and show that head-cycle-free ASPDA programs reduce to the non-disjunctive case—similarly to head-cycle-free ASP programs, but through a more complex transformation. The blowup in the program size is linear in this case.

1 Introduction

Defeasible reasoning is a form of non-monotonic reasoning where logical axioms are true "by default" but their truth status may be undercut or even negated by other, conflicting axioms. This type of reasoning has been an important application of logic programming. It was successfully used to model policies, regulations, and law; actions, change, and process causality; Web services; aspects of inductive/scientific learning and natural language understanding. However, there is a bewildering multitude of dissimilar and incompatible approaches to defeasibility based on a wide variety of intuitions and techniques. The difficulties in relating and comparing the different approaches have been discussed in [13,3] and other works. Combining the various theories of defeasible reasoning with other advances in logic-based knowledge representation, such as HiLog [4] and F-logic [16], has also been a problem.

* This work is part of the SILK (Semantic Inference on Large Knowledge) project sponsored by Vulcan, Inc. It was also partially supported by the NSF grant 0964196.

Our earlier work [20] addressed some of these issues by introducing a general framework for defeasible reasoning, called LPDA, which abstracts the intuitions about defeasibility into what we call *argumentation theories*. This enabled a uniform syntax and semantics for a wide variety of defeasible theories, which could be used in harmony and simultaneously in the same knowledge base. LPDA, as defined in [20], was developed on the basis of the well-founded models [9] and was able to unify a number of approaches to defeasible reasoning that are based on the well-founded semantics. However, a large number of works on defeasible reasoning are based on the stable model semantics [11], which has very different properties and is not capturable by well-founded models. Furthermore, defeasible reasoning in the presence of disjunctive information, which to the best of our knowledge has not been considered hitherto, appears to require even more general semantics, the answer set semantics [10].

The present work takes the idea of LPDA further and introduces ASPDA—an analogous framework for defeasibility of *disjunctive* logic rules through argumentation theories based on Answer Set Programming (ASP). In this way, LPDA and ASPDA together unify and extend most of the existing theories of defeasible reasoning in Logic Programming.

Extension of the semantics of LPDA to ASP with head-disjunctions turned out to be elegant but not straightforward. The relationship between ASPDA and the regular ASP also proved to be non-obvious. First, we show that ASPDA can be expressed by regular ASP programs, albeit at the cost of exponential blowup in the number of rules. Then we study the class of *head-cycle-free* programs with disjunctive heads and show that a related notion exists for ASPDA. By analogy with the classical case, such programs can be reduced to non-disjunctive programs under the defeasible stable model semantics, although the transformation is more complicated than in the case of the regular ASP. The blowup in the program size is still linear, however.

The rest of this paper is organized as follows. Section 2 illustrates defeasible reasoning under the answer-set semantics using the well-known Turkey Shoot example [19]. Section 3 defines the syntax and semantics of defeasible disjunctive logic programs and presents a number of interesting results about reducibility to the regular logic programming and to the non-disjunctive case. Section 4 gives two examples of argumentation theories for ASPDA. One is an adaptation of GCLP [14,20] to ASPDA, a theory that is used in all examples throughout this paper. Another is an argumentation theory that captures Defeasible Logic [1]. Although Defeasible Logic (as all other theories of defeasible reasoning up until now) does not support head-disjuncts in the rules, it is an apt illustration of ASPDA as a unifying framework that is capable of capturing much of the prior work. Sections 5 and 6 discuss related work and conclude the paper.

2 Motivating Example

The following example is adapted from the Texas Turkey Shoot game example in [19]. We use the usual syntax of logic programming with the only difference that rules are tagged with `@tag` symbols and disjunctions are allowed. Variables are prefixed with the symbol `?`. Initially one of the guns is known to be loaded, but it is not known which. The objective is to find a plan to kill the turkey by shooting one or both guns assuming that the shooter can observe the effects of his actions. Let `g1` and `g2` be the constants representing the guns. Numerals are used in the example to represent time points, and the initial time point is assumed to be 1. For instance, `shoot(g1,1)` and `shoot(g1,2)` represent the actions of shooting the gun `g1` at time points 1 and 2.

```
@kpld    loaded(?Gun,?Time+1) :- loaded(?Gun,?Time).    // Frame axiom 1.
@kpunld  neg loaded(?Gun,?T+1) :- neg loaded(?Gun,?T). // Frame axiom 2.
@dd      neg alive(?Time+1) :- neg alive(?Time).       // Frame axiom 3.
@liv     alive(?Time+1) :- alive(?Time).               // Frame axiom 4.
// A gun becomes unloaded after being fired
@sht1    neg loaded(?Gun,?Time+1) :- shoot(?Gun,?Time).
// The turkey becomes dead after a loaded gun is fired at it
@sht2    neg alive(?Time+1) :- shoot(?Gun,?Time), loaded(?Gun,?Time).
//  Axioms for the initial state
         alive(1).                              // The turkey is alive initially
@unld    neg loaded(g1,1) ∨ neg loaded(g2,1). // One gun is unloaded initially
@ld      loaded(g1,1) ∨ loaded(g2,1).        // One gun is loaded initially
         shoot(g1,1).                        // Fire g1 at time 1
         // If g1 is unloaded at time 1, fire g2 at time 2.
         shoot(g2,2) :- not loaded(g1,1).
// axioms for contradiction and rule priorities
#opposes(alive(?Time), neg alive(?Time)).
#overrides(sht1, kpld).
#overrides(sht2, liv).
```

In the above specification, some of the rules have tags, e.g., `kpld` and `sht1`, and the predicate `#overrides` specifies priorities among some of these tagged rules. We distinguish between the classical-logic-like *explicit negation* `neg` and the *default negation* `not` (which in this paper will have the answer-set semantics). Literals L and `neg` L are assumed to be incompatible and cannot both appear in a consistent model. The predicate `#opposes` specifies additional contradictions, such as the inability for the turkey to be both dead and alive at the same time.

We can now explain how defeasible reasoning works in the above game. The rule labeled `kpld` is a frame persistence axiom stating that a loaded gun stays loaded unless some other action explicitly changes this state of affairs. The rule `sht1` states that if a gun is fired then it becomes unloaded in the next state. This rule has a higher priority than the frame axiom `kpld` due to the axiom `#overrides`(sht1,kpld). The rule labeled `liv` is another frame axiom stating that a live turkey remains alive by default. This rule is defeated by the higher-priority rule labeled `sht2`, which says that if a loaded gun is fired at the turkey, then the turkey is dead in the next state. Note that our program has disjunctions in the heads of the rules labeled `unld` and `ld`), so the initial state of the game is uncertain. The problem is to infer that by firing one or both guns in succession the shooter can kill the turkey despite the uncertainty in the initial state. Note that due to the disjunctions, the other existing logic programming approaches to defeasible reasoning cannot handle the above situation, and this is precisely the motivation for our current work. We will return to this example at the end of Section 4.1 after the necessary theory is developed.

3 Defeasible Reasoning with Argumentation Theories

In this section we introduce the syntax and semantics of disjunctive logic programming where defeasibility is controlled by argumentation theories. The main syntactic difference is that rules now have *tags*, and the main semantic difference is that these rules can be *defeated*.

Let $\mathcal{L}$ be a logic language with the usual connectives $\wedge$ for conjunction, $\vee$ for disjunction, and `:-` for rule implication; and two negation operators: `neg` for explicit negation and `not` for default negation. The alphabet of the language consists of: an infinite set of variables, which are shown in the examples as alphanumeric symbols prefixed with the question mark ?; and a set of constant symbols, which can appear as individuals, function symbols, and predicates. Constants will be shown as alphanumeric symbols that are not prefixed with a "?". We assume that the language includes two special propositional constants, $\mathbf{t}$ and $\mathbf{f}$, which stand for *true* and *false*, respectively. We also assume the following order on these propositions: $\mathbf{f} < \mathbf{t}$.

We use the standard notion of *terms* in logic programming. ***Atomic formulas***, also called ***atoms***, can be quite general in form: they can be the usual atoms used in ordinary logic programming; or the higher-order expressions of HiLog [4]; or the frames of F-logic [16]. A ***literal*** has one of the following forms:

– An atomic formula.
– `neg` A and `not` A, where A is an atomic formula.
– `not neg` A, where A is an atomic formula.
– `not not` L and `neg neg` L, where L is a literal; these are identified with L.

For convenience, the literals `not not` L and `neg neg` L will be identified with L. Let A denote an atom. Literals of the form A or `neg` A (or literals that reduce to these forms after elimination of double negation) are called **`not`-free literals**; literals that reduce to the form `not` A are called **`not`-literals**.

Definition 1 (Tagged rule). *A **tagged rule** in a logic language $\mathcal{L}$ is an expression of the form*

$$@r\ L_1 \vee ... \vee L_k \texttt{ :- } Body \tag{1}$$

*where r is a term, called the **tag** of the rule; L_1, ..., L_k ($k \geq 0$) are literals in $\mathcal{L}$, called the **head literals** of the rule; and Body, called the **body** of the rule, is a conjunction of literals in $\mathcal{L}$.[1] As is common in logic programming, we will often write A, B to represent the conjunction $A \wedge B$. A rule tag is not a rule identifier: several rules can have the same tag.*

*A **constraint** is a special form of rule where $\mathbf{f}$ is a single head literal. We will usually omit $\mathbf{f}$ in such rules.*

[1] This is easy to generalize to allow Lloyd-Topor extensions [18].

*A **formula** is a literal, a Boolean combination of literals using conjunction and disjunction, or a rule.* □

We will often omit showing rule tags when they are immaterial.

Definition 2 (Rule handle). *Given a rule of the form (1), the terms of the form*

$$\texttt{handle}(r, L_i), \text{ where } i = 1, ..., k$$

*are called the **handles** for that rule. Here* `handle` *is a binary function symbol specifically reserved for representing rule handles. However, we do not make further assumptions about this symbol.* □

Definition 3 (Ground terms and rules). *A **ground term** is a term that contains no variables, a **ground literal** is a variable-free literal, and a **ground rule** is a rule that has no variables.* □

Definition 4 (ASPDA). *An **answer-set program with defaults and argumentation theories** (an **aspda**, for short) in a logic language $\mathcal{L}$ is a set of tagged rules in $\mathcal{L}$, which can be **strict** or **defeasible**. Sets or rules that do not have disjunctions in the head will be called non-disjunctive* `aspda`*s.* □

Strict rules are used as *definite* statements about the world. In contrast, defeasible rules represent *defeasible defaults* whose instances can be "defeated" by other rules. Inferences produced by the defeated rules are "overridden."

We assume that the distinction between strict and defeasible rules is specified in some way: either syntactically or by means of a predicate. For instance, in Section 4, we use the predicate `#strict` for that purpose.

Aspdas are used in conjunction with *argumentation theories*, which are sets of rules that defines conditions under which some rule instances may be defeated by other rules.

Definition 5 (Argumentation theory). *Let $\mathcal{L}$ be a logic language. An **argumentation theory** is a set, AT, of* strict *rules in $\mathcal{L}$ of the form (1). We also assume that the language $\mathcal{L}$ includes a unary predicate,* $\texttt{\$defeated}_{AT}$*, which may appear in the heads of some rules in AT.*[2] *When confusion does not arise, we will omit the subscript AT.*
An `aspda` *$\mathcal{P}$ is said to be **compatible** with AT if* $\texttt{\$defeated}_{AT}$ *does not appear in the rule heads in $\mathcal{P}$.* □

In argumentation theory all rules are strict, by definition.[3] The rules in AT will normally contain other predicates, besides $\texttt{\$defeated}_{AT}$, that are used to specify how the rules in $\mathcal{P}$ get defeated.

[2] If `$defeated` does not occur in the head of any rule then the semantics of ***aspda***s reduce to ordinary logic programming.

[3] In principle, we could allow argumentation theories to be defeasible, but we will not do so in this paper.

Usually argumentation theories employ the concepts of rule priority and contradictions among facts. Priorities are often specified via a predicate, such as `#overrides`, which tell that some rules (or rule instances) have higher priorities than other rules (e.g., `#overrides`$(rule\_tag1, rule\_tag2)$). Contradictions are commonly expressed via predicates such as `#opposes`, which tell that certain facts cannot be true together (e.g., `#opposes`$(price(ball1, 20), price(ball1, 30))$. The `$defeated` predicate is then defined in terms of these and other predicates. In this paper, we adopt the convention that the predicates defined only by argumentation theories will be prefixed with the \$-sign, the predicates used and/or defined both by the argumentation theories and user programs will be prefixed with the #-sign, and the predicates defined only by user programs will not be marked in any special way: they will be denoted by alphanumeric symbols.

In defining the semantics, we assume that the argumentation theories are ground. A grounded version of AT with respect to a compatible ***aspda*** $\mathcal{P}$ is obtained by appropriately instantiating the variables and meta-predicates.

Note that the theory developed here permits different subsets of the overall ***aspda*** to have different argumentation theories AT with different `$defeated`$_{AT}$ predicates. For instance, our implementation of the logic programming framework with argumentation theories for the well-founded semantics in an extended version of FLORA-2 [15] supports multiple argumentation theories.

3.1 Interpretations and Models

Definition 6 (Herbrand universe). *Let $\mathcal{P}$ be an* `aspda` *and AT an argumentation theory over language $\mathcal{L}$.*

– *The **Herbrand universe** of $\mathcal{P}$, denoted $\mathcal{U}_{\mathcal{L}}$, is the set of all ground terms built using the constants and function symbols that appear in $\mathcal{L}$. When confusion does not arise, we will simply write $\mathcal{U}$.*
– *The **Herbrand base** of $\mathcal{P}$, denoted $\mathcal{B}_{\mathcal{L}}$ (or simply $\mathcal{B}$, when no ambiguity arises), is the set of all ground* `not` *-free literals that can be constructed using the predicates in $\mathcal{L}$.* □

Definition 7 (Herbrand interpretation). *A **Herbrand interpretation, $\boldsymbol{I}$,** is a subset of $\mathcal{B}$. It is simply a set of ground* `not` *-free literals. In addition, $\boldsymbol{I}$ must contain* **t** *and it must not contain* **f**.
*An interpretation is **inconsistent relative to** an atom A if both A and* `neg` *A are in $\boldsymbol{I}$. Otherwise, $\boldsymbol{I}$ is **consistent relative to** A. An interpretation is **consistent** if it is consistent relative to every atom and **inconsistent** if it is inconsistent relative to some atom.* □

Note that all interpretations considered in this paper are Herbrand, so we will often neglect to mention Herbrandness explicitly.

Next we introduce the notion of satisfaction of defeasible rules and strict rules by interpretations.

Definition 8 (Truth valuation). *Let $\boldsymbol{I}$ be a Herbrand interpretation, L a ground* `not` *-free literal, and let F, G be ground formulas. We define **truth valuations** that map formulas to* $\{\mathbf{t},\mathbf{f}\}$ *as follows:*

– $\boldsymbol{I}(L) = \mathtt{t}$ *iff* $L \in \boldsymbol{I}$, $\boldsymbol{I}(L) = \mathtt{f}$ *iff otherwise.*
– $\boldsymbol{I}(\mathtt{not}\, L) = \sim \boldsymbol{I}(L)$*, where* $\sim \mathtt{t} = \mathtt{f}$ *and* $\sim \mathtt{f} = \mathtt{t}$*.*
– $\boldsymbol{I}(F \wedge G) = \mathtt{min}(\boldsymbol{I}(F), \boldsymbol{I}(G))$*. Recall that* $\mathtt{f} < \mathtt{t}$*.*
– $\boldsymbol{I}(F \vee G) = \mathtt{max}(\boldsymbol{I}(F), \boldsymbol{I}(G))$*.*
– *For a strict rule* $@r\ F \,{:\!-}\, G$*, we define* $\boldsymbol{I}(F \,{:\!-}\, G) = \mathtt{t}$ *if and only if* $\boldsymbol{I}(F) \geq \boldsymbol{I}(G)$*.*
– *For a defeasible rule* $@r\ F \,{:\!-}\, G$*, we define* $\boldsymbol{I}(@r\ F \,{:\!-}\, G) = \mathtt{t}$ *if and only if* $\boldsymbol{I}(F) \geq \mathtt{min}(\boldsymbol{I}(G), \boldsymbol{I}(\mathtt{not}\,\mathtt{\$defeated}(\mathtt{handle}(r, F))))$*.*
Here $\mathtt{handle}(r, F)$ *is the handle for the rule* $@r\ F \,{:\!-}\, G$ *(Definition 2).* □

Definition 9 (Model of formula). *If F is a ground formula, $\boldsymbol{I}$ an interpretation, and $\boldsymbol{I}(F) = \mathtt{t}$, then we write $\boldsymbol{I} \models F$ and say that $\boldsymbol{I}$ is a **model** of F or that F is **satisfied** in $\boldsymbol{I}$. An interpretation $\boldsymbol{I}$ is a model of an* aspda *$\mathcal{P}$ if all the rules in $\mathcal{P}$ are satisfied in $\boldsymbol{I}$, i.e., if $\boldsymbol{I} \models R$ for every $R \in \mathcal{P}$.* □

Definition 10 (Model of ASPDA). *Given an* aspda *$\mathcal{P}$, an argumentation theory AT, and an interpretation $\boldsymbol{M}$, we say that $\boldsymbol{M}$ is a model of $\mathcal{P}$ with respect to the argumentation theory AT (or a model of $(\mathcal{P}, AT)$, for short), written as $\boldsymbol{M} \models (\mathcal{P}, AT)$, if $\boldsymbol{M} \models \mathcal{P}$ and $\boldsymbol{M} \models AT$.* □

Definition 11 (Minimal model). *An interpretation $\boldsymbol{M}$ is a minimal model of $(\mathcal{P}, AT)$ iff $\boldsymbol{M}$ is a model of $(\mathcal{P}, AT)$ and no proper subset of $\boldsymbol{M}$ is a model of $(\mathcal{P}, AT)$.* □

3.2 Stable Model and Answer-Set Semantics

In this section, we extend the stable model semantics [11] and the answer-set semantics [10] to ASPDA. We start with non-disjunctive ***aspda***s and stable models.

Definition 12 (ASPDA quotient, non-disjunctive case). *Let $\mathcal{Q}$ be a non-disjunctive* aspda*, and let $\boldsymbol{J}$ be a Herbrand interpretation for $\mathcal{Q}$. The **ASPDA quotient of $\mathcal{Q}$ by $\boldsymbol{J}$**, written as $\dfrac{\mathcal{Q}}{\boldsymbol{J}}$, is defined by the following sequence of steps:*

1. *Delete every rule $R \in \mathcal{Q}$ such that there is a* not *-literal of the form* not *A in R's body and $A \in \boldsymbol{J}$;*
2. *Delete every defeasible rule of the form $@r\ L \,{:\!-}\, Body$ in $\mathcal{Q}$ such that* $\mathtt{\$defeated}(\mathtt{handle}(r, L)) \in \boldsymbol{J}$*.*
3. *Remove all* not *-literals from the remaining rules.*
4. *Remove all tags from the remaining tagged rules.* □

When dealing with stable models, it is often assumed that interpretations are consistent [10]. All the definitions and results in this section extend to this case straightforwardly.

Recall that the *minimality* of Herbrand models is defined in Definition 11.

Definition 13 (Stable model). *A Herbrand interpretation $\boldsymbol{M}$ is a **stable model** of a non-disjunctive* aspda *$\mathcal{P}$ with respect to the argumentation theory AT, if $\boldsymbol{M}$ is a minimal Herbrand model of $\frac{\mathcal{P} \cup AT}{\boldsymbol{M}}$.* □

The next theorem shows that non-disjunctive *aspda*s can be implemented using ordinary logic programming systems that support the stable model semantics (e.g., DLV [17]).

Theorem 1 (Reduction for the stable model semantics). *Let $\mathcal{P}$ be a non-disjunctive* aspda *and AT an argumentation theory. Then the following two sets coincide:*

- *The set of stable models of $\mathcal{P}$ with respect to AT.*
- *The set of stable models of the ordinary logic program $\mathcal{P}' \cup AT'$, where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by converting every defeasible rule* $(\texttt{@r L :- Body}) \in \mathcal{P}$ *into the plain rule of the form* `L :- Body, not $defeated(handle(r, L))` *and removing all the remaining tags; and AT' is obtained from AT by simply removing all the tags.* □

For disjunctive rules, stable models are called *answer sets* and we will now generalize the above semantics to such rules. In generalizing *aspda*s to disjunctive rules, the main problem is to define handles for disjunctive rules, to define quotients, and to find an analog of the reduction theorem.

Example 1. Consider a disjunctive program

```
@r1 a ∨ b ∨ c.
@r2 d ∨ e.
```

The ordinary stable models of this program are {a, d}, {a, e}, {b, d}, {b, e}, {c, d}, and {c, e}. Suppose now that a cannot be true when either d or e holds, and that b, e are also incompatible. We express this with the following facts:

```
#opposes(a,d).        #opposes(a,e).        #opposes(b,e).
```

Suppose, in addition, that rule r1 has a higher priority than r2, i.e.,

```
#overrides(r1,r2).
```

Intuitively, {a, d}, {a, e}, and {b, e} can no longer be models due to the incompatibility statements above, while the models {b, d}, {c, d}, and {c, e} are still intuitively fine. At the same time, one might feel that {a} should be viewed as a suitable model because r1 overrides r2, a makes r1 true, and a is incompatible with both heads of the rule r2.

As it turns out, {a} may or may not be a defeasible stable model—it all depends on the associated argumentation theory. It would be a stable model of our *aspda* if the argumentation theory had the following rule instances:

```
$defeated(handle(r2, d)) :- #overrides(r1, r2), #opposes(a, d), a.
$defeated(handle(r2, e)) :- #overrides(r1, r2), #opposes(a, e), a.
```
□

The following definitions generalize Definition 12 to disjunctive *aspda*s and make the intuition behind Example 1 precise.

Definition 14 (ASPDA quotient, disjunctive case). *Let $\mathcal{Q}$ be a disjunctive* `aspda`*, and let $\boldsymbol{J}$ be a Herbrand interpretation for $\mathcal{Q}$. We define the ASPDA* ***quotient of** $\mathcal{Q}$ **by** $\boldsymbol{J}$, written as $\dfrac{\mathcal{Q}}{\boldsymbol{J}}$, by the following sequence of steps:*

1. *Delete every rule $R \in \mathcal{Q}$ such that there is a* `not`*-literal of the form* `not` *A in R's body and $A \in \boldsymbol{J}$;*
2. *For every defeasible rule of the form $@\mathtt{r}\ \mathtt{L_1} \vee \ldots \vee \mathtt{L_n}$* `:- Body` *in $\mathcal{Q}$, delete every $\mathtt{L_i}$ such that $\mathtt{\$defeated(handle(r, L_i))} \in \boldsymbol{J}$. If all the $\mathtt{L_i}$'s are deleted, delete the entire rule.*
3. *Remove all* `not`*-literals from the remaining rules.*
4. *Remove all tags from the remaining tagged rules.* □

Definition 13 carries over in a natural way:

Definition 15 (Answer set). *A Herbrand interpretation $\boldsymbol{M}$ is an* ***answer set*** *of a disjunctive* `aspda` *$\mathcal{P}$ with respect to the argumentation theory AT, if $\boldsymbol{M}$ is a minimal Herbrand model of $\frac{\mathcal{P} \cup AT}{\boldsymbol{M}}$.* □

The analog of Theorem 1 is as follows.

Theorem 2 (Reduction for the answer-set semantics). *Let $\mathcal{P}$ be a (disjunctive)* `aspda` *and AT an argumentation theory. Then the following two sets coincide:*

- *The set of answer sets for the* `aspda` *$\mathcal{P}$ with respect to AT.*
- *The set of answer sets for the ordinary logic program $\mathcal{P}' \cup AT'$, where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by converting every defeasible rule $(@\mathtt{r}\ \mathtt{L_1} \vee \ldots \vee \mathtt{L_n}\ \mathtt{:-\ Body}) \in \mathcal{P}$ into a collection of plain rules of the form*

$$\vee_{\mathtt{i} \in \mathtt{K}} \mathtt{L_i}\ \mathtt{:-\ Body} \wedge\ \wedge_{i \in K}\ \mathtt{not\ \$defeated(handle(r, L_i))}$$
$$\wedge\ \wedge_{j \in N-K}\ \mathtt{\$defeated(handle(r, L_j))}.$$

for each subset $K \subseteq N = \{1, \ldots, n\}$ and removing all the remaining tags; and AT' is obtained from AT by simply removing all the tags. □

With the above definitions, it can now be verified that the answer sets for the ***aspda*** in Example 1 are precisely as described there.

3.3 Reduction to the Non-disjunctive Case

In ordinary answer-set programming, certain disjunctive rules can be reduced to the non-disjunctive case via the so-called *shifting* transformation. For example, this transformation would replace the rule `p ∨ q ∨ s :- body` with the rules

```
p :- body, not q, not s.
q :- body, not p, not s.
s :- body, not q, not p.
```

Ben-Eliyahu and Dechter [2] have shown that this is an *equivalence* transformation for so called *head-cycle free* programs.[4] We reproduce that definition below adjusting it for disjunctive ***aspda***s.

[4] The works [7,12] developed similar shifting techniques.

Definition 16. *[2] The **dependency graph** $G_{\mathcal{P}}$, of an* `aspda` *$\mathcal{P}$, is a directed graph where nodes are literals. An edge from L to L' goes iff there is a rule in which L appears positive in the body and L' is a head literal. An* `aspda` *is **head-cycle free (HCF)** iff its dependency graph does not contain directed cycles that connect literals that belong to the head of the same rule.* □

An interesting question is whether an analogous shifting transformation and an equivalence result holds for disjunctive ***aspda***s.

Definition 17. *Let $\mathcal{P}$ be a disjunctive* `aspda`*. The (ordinary) **shifting** of $\mathcal{P}$, written as $shift(\mathcal{P})$, is a non-disjunctive* `aspda` *obtained from $\mathcal{P}$ by replacing each defeasible rule of the form* $(@\mathtt{r}\ \mathtt{L}_1 \vee ... \vee \mathtt{L}_n \ \texttt{:-}\ \mathtt{Body}) \in \mathcal{P}$ *with* `n` *new defeasible rules*

$$\begin{array}{lll} @\mathtt{r}\ \mathtt{L}_1 & \texttt{:-} & \mathtt{Body} \wedge \mathtt{not}\, \mathtt{L}_2 \wedge ... \wedge \mathtt{not}\, \mathtt{L}_n \\ ... & ... & ... \\ @\mathtt{r}\ \mathtt{L}_n & \texttt{:-} & \mathtt{Body} \wedge \mathtt{not}\, \mathtt{L}_1 \wedge ... \wedge \mathtt{not}\, \mathtt{L}_{n-1} \end{array}$$ □

Surprisingly, it turns out that $shift(\mathcal{P})$ is *not* equivalent to $\mathcal{P}$ even for HCF ***aspda***s. To see this, consider the following rule set, which we will denote $\mathcal{P}_1$

```
@r1  a ∨ b ∨ c.            @r2  d.            @r3  c.
```

Suppose that the associated argumentation theory implies `$defeated(handle(r1, c))` and does not imply any other `$defeated` facts that involve the above rules. Then $\mathcal{P}_1$ would have the following answer sets: $\{\mathtt{a}, \mathtt{d}, \mathtt{c}\}$ and $\{\mathtt{b}, \mathtt{d}, \mathtt{c}\}$. In contrast, the ordinary shifting transformation yields the non-disjunctive ***aspda*** $shift(\mathcal{P}_1)$

```
@r1  a :-  not b ∧ not c.          @r2  d.
@r1  b :-  not a ∧ not c.          @r3  c.
@r1  c :-  not a ∧ not b.
```

which has only one answer set: $\{\mathtt{d}, \mathtt{c}\}$ with respect to the argumentation theory.

It turns out, however, that a result similar to Ben-Eliyahu and Dechter's holds for disjunctive ***aspda***s, but for a slightly different shifting transformation.

Definition 18. *The **ASPDA shifting** of an* `aspda` *$\mathcal{P}$, written as* `aspda_`*$shift(\mathcal{P})$, is a non-disjunctive* `aspda` *obtained from $\mathcal{P}$ by replacing each defeasible rule of the form* $(@\mathtt{r}\ \mathtt{L}_1 \vee ... \vee \mathtt{L}_n \ \texttt{:-}\ \mathtt{Body}) \in \mathcal{P}$ *with* `n` *new defeasible rules of the form*

$$\begin{array}{lll} @\mathtt{r}\ \mathtt{L}_1 & \texttt{:-} & \mathtt{Body} \wedge \big(\mathtt{not}\, \mathtt{L}_2 \vee \mathtt{\$defeated}(\mathtt{handle}(\mathtt{r}, \mathtt{L}_2))\big) \\ & & \wedge \ \ ... \\ & & \wedge \big(\mathtt{not}\, \mathtt{L}_n \vee \mathtt{\$defeated}(\mathtt{handle}(\mathtt{r}, \mathtt{L}_2))\big) \\ ... & & ... \qquad ... \\ @\mathtt{r}\ \mathtt{L}_n & \texttt{:-} & \mathtt{Body} \wedge \big(\mathtt{not}\, \mathtt{L}_1 \vee \mathtt{\$defeated}(\mathtt{handle}(\mathtt{r}, \mathtt{L}_1))\big) \\ & & \wedge \ \ ... \\ & & \wedge \big(\mathtt{not}\, \mathtt{L}_{n-1} \vee \mathtt{\$defeated}(\mathtt{handle}(\mathtt{r}, \mathtt{L}_{n-1}))\big) \end{array} \qquad (2)$$

Theorem 3. *Let $\mathcal{P}$ be an HCF* `aspda` *and let AT be an argumentation theory. Then S is an answer set of $\mathcal{P}$ with respect to AT iff S is an answer set of* `aspda`$\_shift(\mathcal{P})$ *with respect to AT.* □

Corollary 1. *Let $\mathcal{P}$ be an HCF* `aspda`*. Let AT be an argumentation theory such that, for each literal L, whenever* `$defeated(handle(`r, L`))` *is true for some rule tag r,* `$defeated(handle(`r', L`))` *is true for every tag r' such that there is a rule with the tag r' and L as a head-literal. Then S is an answer set of* `aspda`$\_shift(\mathcal{P})$ *with respect to AT iff S is an answer set of $shift(\mathcal{P})$. In other words,* `aspda`$\_shift(\mathcal{P})$ *is equivalent to $shift(\mathcal{P})$ with respect to AT.*

4 Examples of Argumentation Theories

4.1 A-GCLP [14,20]

Our first example is an ASPDA counterpart for the argumentation theory proposed in [20], which captures generalized courteous logic programs [14] (under the well-founded semantics). We will call this theory A-GCLP and will denote it by AT^{AGCLP}. It is this argumentation theory that was used in all the earlier examples in this paper.

In AT^{AGCLP}, the predicate `$defeated`, which plays a key role in the semantics of ***aspdas***, is defined in terms of the predicates `#opposes` and `#overrides`. These predicates are defined by the knowledge engineer within the knowledge base via sets of facts and rules. The argumentation theory only imposes some constraints on `#opposes`.

The `$defeated` predicate is now defined as follows. A rule handle is *defeated* if it is *refuted* by some other not defeated rule or if it transitively defeats itself.

```
$defeated(?R) :- $defeats(?S,?R).
$defeated(?R) :- $trans_defeats(?R,?R).
```

The auxiliary predicates used above are defined as follows:

```
$defeats(?R,?S) :- $refutes(?R,?S), not $defeated(?R), not #strict(?S).
$trans_defeats(?X,?Y) :- $defeats(?X,?Y).
$trans_defeats(?X,?Y) :- $defeats(?X,?Z), $trans_defeats(?Z,?Y).
```

The predicate `#strict` is used here to distinguish strict rules from the defeasible ones. The predicate `$refutes` indicates when one rule handle refutes another. *Refutation* of a rule handle, `r`, means that a higher-priority rule implies a conclusion that is incompatible with the conclusion implied by the rule with the handle `r`. This is defined as follows:

```
$refutes(?R,?S) :-
        $conflict(?R,?S), #overrides(?R,?S), ?R = handle(?T,?L), ?L.
```

The definition of the concept of a conflict between two rules, represented by the predicate `$conflict` above, relies in turn on the notion of a candidate. A *candidate* rule-instance is one whose body is true in the knowledge base:

```
$candidate(?R) :- body(?R,?B), ?B.
```

Here the meta-predicates **body** binds $?B$ to the body of a rule with handle $?R$.

Conflicting rules are now defined as follows: two rule handles are in conflict if they are both candidates and the literals in them are incompatible:

$$\begin{aligned}&\texttt{\$conflict}(?R1, ?R2)\texttt{ :- }\\ &\qquad ?R1 = \texttt{handle}(?T1, L1),\ ?R2 = \texttt{handle}(?T2, L2),\\ &\qquad \texttt{\$candidate}(?R1),\ \texttt{\$candidate}(?R2),\ \texttt{\#opposes}(?L1, ?L2).\end{aligned} \tag{3}$$

Finally, the argumentation theory provides the following self-explanatory background axioms for `#opposes`, and the axiom of preference for strict rules:

$$\begin{aligned}&\texttt{\#opposes}(?L1, ?L2)\ \texttt{:-}\ \texttt{\#opposes}(?L2, ?L1).\\ &\texttt{\#opposes}(?L,\ \texttt{neg}\ ?L).\\ &\texttt{:-}\ ?L1,\ ?L2,\ \texttt{\#opposes}(?L1, ?L2).\\ &\texttt{\#overrides}(?R, ?S)\ \texttt{:-}\ \texttt{\#strict}(?R),\ \texttt{not}\,\texttt{\#strict}(?S).\end{aligned}$$

With this argumentation theory, we can now come back to the turkey-shoot example in Section 2 and to Example 1. It can be verified that the turkey-shoot example has two answer sets. In one, {`neg loaded(g1, 1)`, `loaded(g2, 1)`, `neg alive(3)`} is true and in another {`loaded(g1, 1)`, `neg loaded(g2, 1)`, `neg alive(3)`}. This shows that the sequence of actions in the example produces the expected result and AT^{AGCLP} allows us to reason by cases.

As to Example 1, one can verify that this ***aspda*** has four answer sets: {`a`}, {`b`, `d`}, {`c`, `d`}, {`c`, `f`}, as claimed.

4.2 Defeasible Logic [1]

As yet another example, this section develops an argumentation theory that captures the reasoning in Defeasible Logic of [1].

Defeasible Logic partitions all rules into strict, defeasible, and defeaters. The defeater rules are used only to defeat other rules, but they themselves do not produce any inferences. In our terms, this means that defeater rules are *defeated* defeasible rules whose only purpose is to block inferences produced by other rules. Strict and defeater rules are specified via the predicates `#strict` and `#defeater`. Other key restrictions in that logic are that it does not support disjunctions in the rule heads; opposition among literals is limited to p and `neg`p, for each p; does not use default negation, i.e., all literals in that logic are `not`-free; and the rule tags are also rule identifiers. This implies that each tag uniquely determines the rule head and body and this restriction lets us simplify the argumentation theory by omitting `handle` from most literals.

With this, we can now formulate the argumentation theory for Defeasible Logic, denoted AT^{DL}, as follows.

$$\begin{aligned}&\texttt{\$defeated}(\texttt{handle}(?T, ?L))\ \texttt{:-}\ \texttt{\#defeated\_aux}(?T).\quad \textit{// no handles in } AT^{DL}\\ &\texttt{\#defeated\_aux}(?T)\ \texttt{:-}\ \texttt{\$conflict}(?T, ?S),\ \textbf{head}(?S, ?L),\ \texttt{\$definitely}(?L).\\ &\texttt{\#defeated\_aux}(?T)\ \texttt{:-}\ \texttt{\#defeater}(?T).\qquad \textit{// defeaters make no inferences}\\ &\texttt{\#defeated\_aux}(?T)\ \texttt{:-}\ \texttt{\$overruled}(?T).\end{aligned}$$

Here **head** is a meta-predicate that binds $?L$ to the head of a rule with Id $?S$.

The predicate `$definitely` is defined as follows:

```
$definitely(?L) :-
        #strict(?T), head(?T,?L), body(?T,?B), each_definite(?B).
```

As before, **body** is a meta-predicate that binds $?B$ to the body of a rule with tag $?T$; **each_definite**($?B$) is a meta predicate that is true when `$definitely`($?B$) is true or when $?B$ is bound to a conjunction, *conj*, and `$definitely`(c) is true for every conjunct $c \in conj$.

It remains to define `$overruled`, which relies on the notion of candidacy and conflict, like in AT^{AGCLP}. The predicate `$candidate` is defined as in ([3]) except that the handles are dropped and only the rule tags are retained.

```
  $overruled(?T) :- $conflict(?T,?S), $candidate(?S), not $refuted(?S).
    $refuted(?S) :- $conflict(?T,?S), $candidate(?T),
                    #overrides(?T,?S), not #defeater(?T).
$conflict(?T,?S) :- head(?T,?L), head(?S,neg ?L),
                    $candidate(?T), $candidate(?S).
```

5 Comparison with Other Work

Although a great deal of work has been devoted to various theories of defeasible reasoning, only a few dealt with unifying frameworks for such reasoning. The notable exceptions are the works [13,5,8], which had goals similar to ours. Due to the large volume of literature on defeasible reasoning, we will focus on the above works, which are related to our work most closely. We refer the reader to a recent survey [6] for a discussion of the various individual theories of defeasibility.

The logic of prioritized defaults [13] does not use the notion of argumentation theories, but it allows for multiple theories of defaults for different application domains. This is analogous to allowing argumentation theories to vary. However, defaults are defined via meta-theories and the semantics in [13] is given by meta-interpretation. What we call an "argumentation theory" is implicit in the meta-interpreters, and no independent model theory is given. In contrast, our approach abstracts all the differences between the different theories for defaults to the notion of an argumentation theory with a simple interface to the user-provided domain description, the predicate `$defeated`. Our approach is model-theoretic and it covers both the well-founded semantics [20] and answer sets (this work). It unifies the theories of Courteous Logic Programming, Defeasible Logic, Prioritized Defaults, and more.

Delgrande et. al. [5] propose a framework of ordered logic programming, which can use a variety of preference handling strategies. For each strategy, this approach devises a transformation from ordered logic programs to ordinary logic programs. Each transformation is custom-made for the particular preference-handling strategy, and the approach was illustrated by showing transformations for several strategies, including two described in earlier works [21,8].

Unlike ASPDA, Delgrande's framework does not come with a unifying model-theoretic semantics. Instead, the definition of preferred answer sets differs from one preference-handling strategy to another. One of the more important conceptual differences between our work and [5] has to do with the nature of the variable parts of the two approaches. In our case, the variable part is the argumentation theory, which is a set of definitions for concepts that a human reasoner might use to argue why certain conclusions are to be defeated. In case of [5], the variable part is the transformation, which encodes a fairly low-level mechanism: the order of rule applications required to generate the preferred answer set.[5] It is also important to note that each program transformation in [5] needs a compiler that contains hundreds of lines of Prolog code, while our approach requires no new software, and each argumentation theory typically contains 20-30 rules.

Leone et. al. [8] set out to unify approaches to defeasible reasoning. Specifically, they present an adaptable meta-interpreter, which can be made to simulate the approaches described in [3,21] among others. However, this framework lacks a model theoretic semantics and is not as flexible as ASPDA.

Finally, to the best of our knowledge, the present paper is the only work that studies the semantics of defeasibility for *disjunctive* logic programs.

6 Conclusions

This paper developed a novel theory of defeasible disjunctive logic programming under the answer-set semantics. It is a companion to our earlier work which developed a general theory of defaults and defeasibility through argumentation theories and was based on the well-founded semantics. Apart from the model theoretic semantics, we have shown that head-cycle free disjunctive defeasible programs can be reduced to non-disjunctive ones, which mirrors an analogous result for non-defeasible disjunctive rules with default negation. To illustrate the power of the proposed framework, we have given two examples of argumentation theories. One is an adaptation for stable models of the generalized courteous argumentation theory, which was presented in [20] for well-founded models. This theory was used in all the examples in this paper. The other argumentation theory was intended to show how $ASPDA$ can capture other approaches to defeasible reasoning; in this case the defeasible logic of [1].

References

1. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Embedding defeasible logic into logic programming. Theory and Practice of Logic Programming (TPLP) 6(6), 703–735 (2006)
2. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Ann. Math. Artif. Intell. 12(1-2), 53–87 (1994)
3. Brewka, G., Eiter, T.: Prioritizing default logic. In: Intellectics and Computational Logic – Papers in Honour of Wolfgang Bibel, pp. 27–45. Kluwer Academic Publishers, Dordrecht (2000)

[5] Note that argumentation theories can also encode rule application orderings.

4. Chen, W., Kifer, M., Warren, D.: HiLog: A foundation for higher-order logic programming. Journal of Logic Programming 15(3), 187–230 (1993)
5. Delgrande, J., Schaub, T., Tompits, H.: A framework for compiling preferences in logic programs. Theory and Practice of Logic Programming 2, 129–187 (2003)
6. Delgrande, J., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. Computational Intelligence 20(12), 308–334 (2004)
7. Dix, J., Gottlob, G., Marek, V.: Reducing disjunctive to non-disjunctive semantics by shift-operations. Fundamenta Informaticae XXVIII(1/2), 87–100 (1996)
8. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Computing preferred answer sets by meta-interpretation in answer set programming. Theory and Practice of Logic Programming 3(4), 463–498 (2003)
9. Gelder, A.V., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM 38, 620–650 (1991)
10. Gelfond, M.: Answer sets. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 285–316. Elsevier, Amsterdam (2008)
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of ICLP/SLP, pp. 1070–1080. MIT Press, Cambridge (1988)
12. Gelfond, M., Przymusinska, H., Lifschitz, V., Truszczynski, M.: Disjunctive defaults. In: Proceedings of the International Conference on Knowledge Representation and Reasoning, pp. 230–237 (1991)
13. Gelfond, M., Son, T.: Reasoning with prioritized defaults. In: Dix, J., Moniz Pereira, L., Przymusinski, T.C. (eds.) LPKR 1997. LNCS (LNAI), vol. 1471, pp. 164–223. Springer, Heidelberg (1998)
14. Grosof, B.: A courteous compiler from generalized courteous logic programs to ordinary logic programs. Technical Report Supplementary Update Follow-On to RC 21472, IBM (July 1999)
15. Kifer, M.: FLORA-2: An object-oriented knowledge base language. The FLORA-2 Web Site, `http://flora.sourceforge.net`
16. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of ACM 42, 741–843 (1995)
17. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
18. Lloyd, J.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987)
19. Morales, A.R., Tu, P.H., Son, T.C.: An extension to conformant planning using logic programming. In: IJCAI, pp. 1991–1996 (2007)
20. Wan, H., Grosof, B., Kifer, M., Fodor, P., Liang, S.: Logic programming with defaults and argumentation theories. In: ICLP, pp. 432–448 (2009)
21. Wang, K., Zhou, L., Lin, F.: Alternating fixpoint theory for logic programs with priority. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 164–178. Springer, Heidelberg (2000)

A Rule-Based Approach to XML Processing and Web Reasoning[⋆]

Jorge Coelho[1], Besik Dundua[2], Mário Florido[2], and Temur Kutsia[3]

[1] ISEP & LIACC, Porto, Portugal
jcoelho@liacc.up.pt
[2] DCC-FC & LIACC, University of Porto, Portugal
{bdundua,amf}@dcc.fc.up.pt
[3] RISC, Johannes Kepler University, Linz, Austria
kutsia@risc.uni-linz.ac.at

Abstract. We illustrate the potential of conditional hedge transformations in Web-related applications on the example of PρLog: an extension of logic programming with advanced rule-based programming features for hedge transformations, strategies, and regular constraints.

1 Introduction

The rule-based approach has been used extensively in many fields, such as expert systems, theorem proving, tree automata, software building and configuration, banking systems, just to name a few. In recent years, the rule-based approach has been experiencing growing popularity in Web applications. One could mention document processing and Web reasoning as prominent examples. The REWERSE project [23] provides an extensive reference material on those topics.

The goal of this paper is to illustrate the potential of strategy-based conditional hedge transformations in Web-related applications. To achieve this goal, first, we present a practical tool: an extension of logic programming with advanced rule-based programming features for hedge transformations, strategies, and regular constraints. Second, we show how it can be used it XML querying, validation, and some basic Web reasoning.

The tool we describe here is PρLog [13] (pronounced Pē-rō-log). It is a Prolog implementation of the ρLog calculus [19], which extends the host language with strategic conditional transformation rules. These rules (basic strategies) define transformation steps on hedges. (A hedge is a sequence of unranked terms.) Strategy combinators help to combine strategies into more complex ones in a declaratively clear way. Transformations are nondeterministic and may yield several results, which fits very well into the logic programming paradigm. Strategic

[⋆] Partially funded by LIACC through Programa de Financiamento Plurianual of the Fundação para a Ciência e Tecnologia (FCT), by the FCT fellowship (ref. SFRH/BD/62058/2009), and by the EC FP6 for Integrated Infrastructures Initiatives under the project SCIEnce (contract No. 026133).

rewriting separates term traversal control from transformation rules. The separation of strategies and rules makes rules reusable in different transformations.

PρLog programs consist of clauses. The clauses either define user-constructed strategies by (conditional) transformation rules or are ordinary Prolog clauses. Prolog code can be used freely within PρLog programs, which is especially convenient when arithmetic calculations or input-output features are needed.

PρLog uses four different kinds of variables, which permits to traverse hedges in single/arbitrary width (with individual and sequence variables) and terms in single/arbitrary depth (with functional and context variables). It provides a possibility to extract an arbitrary subhedge from a hedge, or to extract subterms at arbitrary depth. In addition, PρLog permits regular constraints to restrict possible values of sequence and context variables by regular hedge expressions and regular tree (context) expressions, respectively. These constraints are very useful, for instance, in validation of an XML document with respect to a given DTD.

PρLog has not been implemented specifically for Web-related applications. Its main purpose is to bring strategy-based conditional hedge transformations in the logic programming framework for general programming. The role of PρLog in this paper is to provide a practical platform to illustrate suitability of the calculus behind it in XML querying, validation, and Web reasoning.

In the context of XML processing, the approach PρLog is based on can be classified as positional or pattern-based, where programmer specifies patterns including variables. Examples of such languages are Xcerpt [7], UnQL [8], XDuce [15], and CDuce [4]. Usually, in this approach, variables in patterns specify the nodes to be selected. With PρLog, we can select not only nodes but also sequences of nodes, node labels, and the context around a node that is at arbitrary depth. Moreover, it can naturally express incomplete query patterns.

Approaches to XML, based on the logic programming paradigm, have been quite popular. Besides the already mentioned Xcerpt, the languages like Elog [3], XPathLog [21], and XCentric [11,12] belong to this category. The latter one, like PρLog, represents XML data as an unranked Prolog term and uses sequence matching with regular types for querying. In fact, for our experiments we used XCentric's XML-to-unranked-term translator.

From the general programming point of view, we should mention a number of calculi and languages for rule- and strategy-based programming related to our work, such as rewriting logic [20], ρ-calculus [9], ASF-SDF [25], CHR [14], ELAN [6], Maude [10], the OBJ family of languages [22], Stratego [26], and TOM [2]. ρLog, the calculus behind PρLog, has been influenced by the ρ-calculus. However, there are specific features in ρLog that makes it significantly different from the ρ-calculus: logic programming semantics, top-position matching, hedge transformations, different kinds of variables, and regular constraints.

2 PρLog

In this section we give a brief overview of basic features of PρLog, explaining them mostly on examples instead of giving formal definitions.

Terms and hedges (sequences of terms) in PρLog are built over unranked function symbols and four kinds of variables: individual, sequence, function, and context variables. These sets are disjoint. In this paper we follow the PρLog notation for this language, writing its constructs in `typewriter` font. PρLog uses the following conventions for the variables names: Individual variables start with `i_` (like, e.g., `i_Var` for a named variable or `i_` for the anonymous variable), sequence variables start with `s_`, function variables start with `f_`, and context variables start with `c_`. The function symbols, except the special constant `hole`, have flexible arity. To denote function symbols, PρLog basically follows the Prolog conventions for naming functors, operators, and numbers. Terms `t` and hedges `h` are constructed by the grammars:

```
t ::= i_X | f(h) | f_X(h) | c_X(t)    h ::= t | s_X | eps | (h_1, h_2)
```

where `eps` stands for the empty hedge and is omitted whenever it appears as a subhedge of another hedge. `a(eps)` and `f_X(eps)` are often abbreviated as `a` and `f_X`. A *Context* is a term with a single occurrence of `hole`. A context `C` can be applied to a term `t`, written `C[t]`, replacing the hole in `C` by `t`.

A *substitution* is a mapping from individual variables to hole-free terms, from sequence variables to hole-free hedges, from function variables to function variables/symbols, and from context variables to contexts, such that all but finitely many individual, sequence, and function variables are mapped to themselves, and all but finitely many context variables are mapped to themselves applied to the `hole`. This mapping is extended to terms/hedges in the standard way.

Matching problems are pairs of hedges, one of which is ground (i.e., does not contain variables). Such matching problems may have zero, one, or more (finitely many) solutions, called matching substitutions or *matchers*. For instance, the hedge `(s_1,f(i_X),s_2)` matches `(f(a),f(b),c)` in two different ways: one by the matcher `{s_1↦eps,i_X↦a,s_2↦(f(b),c)}`, the other one by the matcher `{s_1↦f(a),i_X↦b,s_2↦c}`. Similarly, the term `c_X(f_Y(a))` matches the term `f(a,g(a))` with the matchers `{c_X↦f(hole,g(a)),f_Y↦f}` and `{c_X↦f(a,g(hole)),f_Y↦g}`. An algorithm to solve matching problems in the described language has been introduced in [17].

Instantiations of sequence and context variables can be restricted by regular hedge and regular context languages, respectively, specified by the corresponding regular expressions. We do not go into the details of the regular constraints here, just mention that they can be added to matching problems to restrict the set of computed matchers, e.g., matching `c_X(f_Y(a))` to `f(a,g(a))` under the constraint `c_X in f(a,g(hole)`*`)` gives one matcher `{c_X↦f(a,g(hole)),f_Y↦g}` instead of two for the unconstrained case.

A ρLog *atom* (ρ-atom) is a quadruple consisting of a hole-free term `st` (a *strategy*), two hole-free hedges `h1` and `h2`, and a set of regular constraints `R` where each variable is constrained only once, written as `st :: h1 ==> h2 where R`. Intuitively, it means that the strategy `st` transforms `h1` to `h2` when the variables satisfy the constraint `R`. When `R` is empty, we omit it and write `st :: h1 ==> h2`. The negated atom is written as `st :: h1 =\=> h2 where R`. A ρLog *literal* (ρ-literal) is a ρ-atom or its negation. A PρLog *clause* is either a Prolog clause, or a clause of the form `st :: h1 ==> h2 where R :- body` (in the sequel called a ρ-clause) where `body` is a (possibly empty) conjunction of ρ- and Prolog literals.

A PρLog *program* is a sequence of PρLog clauses. A *query* is a conjunction of ρ- and Prolog literals. ρ-clauses and queries can contain only ρLog variables. Prolog clauses and queries can contain only Prolog variables. If a Prolog literal occurs in a ρ-clause or query, it may contain only ρLog individual variables that internally get translated into Prolog variables.

PρLog inference mechanism is based essentially on SLDNF-resolution [1] adapted to ρ-clauses. If the selected literal in the query is a ρ-atom of the form `st :: h1 ==> h2 where R`, then PρLog finds a (renamed copy of a) ρ-clause `st' :: h1' ==> h2' where R' :- body` such that `st'` matches `st` and `h1'` matches `h1` with a substitution σ and the constraint `R'` is satisfied. Then, it replaces the selected literal in the query with the conjunction of `body`σ and a literal `id :: h2'`σ `==> h2 where R`, applies σ to the rest of the query and continues. `id` is a built-in strategy that forces `h2` to match `h2'`σ under `R`. To make sure that in this process we have matching and not unification, we impose *well-modedness* restrictions on ρ-clauses and queries. This is a quite technical notion, whose exact definition can be found in [19]. It guarantees that `h1` and `h2'`σ are ground. Negative ρ-literals are processed by the negation-as-failure rule.

3 XML Processing and Web Reasoning in PρLog

In this section, we illustrate how PρLog can be used in XML querying, validation, and reasoning. PρLog uses the unranked tree model, represented as a Prolog term. Below we assume that the XML input is provided in the translated form.

Querying. In [18], a list of query operations desirable for an XML query language is given: selection, extraction, reduction, restructuring, and combination. They all should be expressible in a single language. A comparison of five query languages on the basis of these queries can be found in [5]. Here we demonstrate, on an example, how selection, extraction, and reduction can be expressed in PρLog. The space limit does not allow us to illustrate the other queries.

Example 1. A car dealer office contains documents from different car dealers and brokers. There are two kinds of documents. The `manufacturer` documents list the manufacturer's name, year, and models with their names, front rating, side rating, and rank. The `vehicle` documents list the vendor, make, model, year, color and price. They are presented by XML data of the following form:

```
<manufacturer>                      <vehicle>
   <mn-name>Mercury</mn-name>          <vendor>
   <year>1998</year>                      Scott Thomason
   <model>                             </vendor>
     <mo-name>Sable LT</mo-name>       <make>Mercury</make>
     <front-rating>3.84                <model>Sable LT</model>
     </front-rating>                   <year>1999</year>
     <side-rating>2.14                 <color>
     </side-rating>                              metallic blue
     <rank>9</rank>                           </color>
   </model> ...                               <price>26800</price>
</manufacturer>                            </vehicle>
```

We assume that sequences of these elements are wrapped respectively by `<list-manuf>` and `<list-vehicle>` tags. To save space, in the queries below we use metavariable M to refer to the document with the root tag `<list-manuf>` and V to the document with the root tag `<list-vehicle>`.

Selection and Extraction: We want to select and extract `<manufacturer>` elements where some `<model>` has `<rank>` less or equal to 10.

```
select_and_extract :: list_manuf(s_,c_Manuf(rank(i_Rank)),s_) ==>
                      c_Manuf(rank(i_Rank)) :-
    i_Rank =< 10.
```

Given the goal `select_and_extract ::` M `==> i_M`, this code generates all solutions, one after the other, via backtracking. The alternatives are generated according how `list_manuf(s_,c_Manuf(rank(i_Rank)), s_)` matches M.

Reduction: From the `<manufacturer>` elements, we want to drop the `<model>` subelements whose `<rank>` is greater than 10. Besides that, we also want to elide the `<front_rating>` and `<side_rating>` elements from the remaining models. It can be done in various ways in PρLog. One of such implementations is given below. `reduction` is defined as the normal form of transforming each `manufacturer` element inside `list_manuf`. A single `manufacturer` element is transformed by `reduction_step` depending whether it contains a model with the rank ≤ 10:

```
reduction :: list_manuf(s_1) ==> list_manuf(s_2) :-
    map1(nf(reduction_step)) :: s_1 ==> s_2,
    !.

reduction_step :: manufacturer(s_1,model(s_,rank(i_R)),s_2) ==>
                  manufacturer(s_1,s_2) :-
    i_R > 10.
reduction_step :: manufacturer(s_1,model(i_Name,i_,s_,rank(i_R)),s_2) ==>
                  manufacturer(s_1,model(i_Name,rank(i_R)),s_2) :-
    i_R =< 10.
```

Here `nf` is the built-in strategy for a normal form computation. Another built-in strategy, `map1`, maps its argument strategy to each single term of the input hedge. The query `reduction ::` M `==> i_List` produces the list of reduced `manufacturer` elements.

Incomplete Queries. Often, the structure of a document to be queried is unknown to a query author, or she is interested not in the entire document but only in its relevant parts. A pattern-based Web querying language should be able to express such incomplete queries. Schaffert in [24] classifies incomplete queries (four kinds of incompleteness: in breadth, in depth, with respect to order and with respect to optional elements) and explains how they are dealt with in Xcerpt. Here we show how they can be expressed in PρLog. As we will see, it can be done pretty naturally, without introducing additional constructs for them.

Incompleteness in breadth. In languages that have wildcards only for single terms, expressing incompleteness in breadth requires a special construct that allows to omit those wildcards for neighboring nodes in the data tree. In PρLog, we do not need any extra construct because of sequence variables. Anonymous sequence variables can be used as wildcards for arbitrary sequence of nodes. Furthermore, if needed, we can use named sequence variables to extract arbitrary sequence of nodes without knowing the exact structure. These are very convenient features, as one can see from the examples in the previous section.

Incompleteness in depth. It allows to select data items that are located at arbitrary, unknown depth and skip all structure in between. For this, in PρLog we just have to place the corresponding query subterm under an anonymous context variable. Moreover, if needed, we can extract the entire context above the query subterm without knowing the structure of the context, by putting there a named context variable. This has been done in the `select_and_extract` clause in the previous section with the `c_Manuf` variable. In fact, that clause also demonstrates how PρLog can combine incompleteness in breadth and depth in a single rule.

Incompleteness with respect to order. It allows to specify neighboring nodes in a different order than the one in that they occur in the data tree. Since PρLog does not permit matching in orderless theories,[1] we need a bit of more coding here. Assume that we do not know in which order the `front_rating` and `side_rating` elements occur in the model in Example 1 and write the clause that extract them:

```
extract_ratings :: c_(model(s_X)) ==> (i_Front,i_Side) :-
    id :: model(s_X) ==> model(s_1,front_rating(i_Front),s_2),
    id :: model(s_1,s_2) ==> model(s_,side_rating(i_Side),s_).
```

In the first subgoal of the body of this rule, the `id` strategy forces the term `model(s_1,front_rating(i_Front),s_2)` to match `model(s_X)`, extracting the value for front rating `i_Front`. Next, to find the side rating, we force matching `model(s_,side_rating(i_Side),s_)` to `model(s_1,s_2)` that is obtained from `model(s_X)` by deleting `front_rating(i_Front)`. This deletion comes for free from the previous match and we can take an advantage of it, since there is no need to keep `front_rating` in the structure where `side_rating` is looked for.

Incompleteness with respect to optional elements. Since seq. variables can be instantiated with the empty hedge, such queries are trivially expressed in PρLog.

Validation. PρLog regular constraints can be used to check whether an XML document conforms to certain DTD that can be expressed by means of regular hedge expressions. We demonstrate it in the following example:

Example 2. Let the DTD below define the structure of the document containing manufacturer elements:

[1] The orderless property is a generalization of commutativity for unranked function symbols. For orderless matching over unranked terms, see [16].

```
<!ELEMENT list-manuf (manufacturer*)>
<!ELEMENT manufacturer (mn-name, year, model*)>
<!ELEMENT model (mo-name, front-rating, side-rating, rank)>
<!ELEMENT mn-name (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT front-rating (#PCDATA)>
<!ELEMENT side-rating (#PCDATA)>
<!ELEMENT rank (#PCDATA)>
```

Then the validation task becomes a PρLog clause where DTD is encoded in a regular constraint:

```
validate :: s_X ==> true
   where [s_X in list_manuf(manufacturer(mn_name(i_),year(i_),
                            model(mo_name(i_),front_rating(i_),
                              side_rating(i_),rank(i_))*)*)]
```

(With `i_` in the constraint we abbreviate the set of all ground terms with respect to the given finite alphabet.) To check whether a certain document conforms this DTD, we take a PρLog term T that represents that document and write the query `validate ::` T `==> true`. The matching algorithm will try to match `s_X` to T and check whether the constraints are satisfied. If the document conforms the DTD, the query will succeed, otherwise it will fail.

Basic Web Reasoning. Semantic Web adds metadata to Web resources, which can be used to make retrieval "semantic". To query both data and metadata, languages need to have certain reasoning capabilities.

Example 3 (Clique of Friends, [24]). This example illustrates some basic reasoning (mainly the transitive closure of a relation) for the Semantic Web. It does not use any particular Semantic Web language itself.

Consider a collection of address books where each address book has an owner and a set of entries, some of which are marked as `friend` to indicate that the person associated with this entry is considered a friend by the owner of the address book. In XML, this collection of address books can be represented in a straightforward manner as follows:

```
<address-books>
 <address-book>
  <owner>Donald Duck</owner>
  <entry>
   <name>Daisy Duck</name>
   <friend/>
  </entry>
  <entry>
   <name>Scrooge McDuck</name>

  </entry>
 </address-book>
 <address-book>
  <owner>Daisy Duck</owner>
  <entry>
   <name>Gladstone Duck</name>
   <friend/>
  </entry>
  <entry>
   <name>Ratchet Gearloose</name>
   <friend/>
  </entry>
 </address-book>
</address-books>
```

The collection contains two address books, the first owned by Donald Duck and the second by Daisy Duck. Donalds address book has two entries, one for Scrooge, the other for Daisy, and only Daisy is marked as friend. Daisys address book again has two entries, both marked as friend.

The *clique-of-friends* of Donald is the set of all persons that are either direct friends of Donald (i.e. in the example above only Daisy) or friends of a friend (i.e. Gladstone and Ratchet), or friends of friends of friends (none in the example above), and so on. To retrieve these friends, we have to define the relation "being a friend of" and its transitive closure.

Transitive closure of a relation can be easily defined in PρLog. It can be even written in a generic way, parameterized by the strategy that defines the relation:

```
transitive_closure(i_Strategy) :: s_X ==> s_Y :-
     i_Strategy :: s_X ==> s_Y.
transitive_closure(i_Strategy) :: s_X ==> s_Z :-
     i_Strategy :: s_X ==> s_Y,
     transitive_closure(i_Strategy) :: s_Y ==> s_Z.
```

The relation of "being a friend of" with respect to the address books document is defined as follows:

```
friend_of(address_books(s_,
             address_book(owner(i_X),s_,entry(name(i_Y),friend),s_),
             s_)) :: i_X ==> i_Y.
```

The query transitive_closure(friend_of(T)) :: Donald_Duck ==> i_Y, where T is the PρLog term corresponding to the address book XML document above, will return one after the other the friend and the friends of the friend of Donald_Duck: Daisy_Duck, Gladstone_Duck, and Ratchet_Gearloose.

References

1. Apt, K.R., Bol, R.: Logic programming and negation: A survey. J. Logic Programming 19, 9–71 (1994)
2. Balland, E., Brauner, P., Kopetz, R., Moreau, P.-E., Reilles, A.: Tom: Piggybacking rewriting on java. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 36–47. Springer, Heidelberg (2007)
3. Baumgartner, R., Flesca, S., Gottlob, G.: The Elog web extraction language. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 548–560. Springer, Heidelberg (2001)
4. Benzaken, V., Castagna, G., Frisch, A.: CDuce: an XML-centric general-purpose language. In: Proc. ICFP'03, pp. 51–63. ACM, New York (2003)
5. Bonifati, A., Ceri, S.: Comparative analysis of five XML query languages. ACM SIGMOD Record 29(1), 68–79 (2000)
6. Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P.-E., Vittek, M.: Elan: A logical framework based on computational systems. ENTCS 4 (1996)
7. Bry, F., Schaffert, S.: Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In: Stuckey, P.J. (ed.) ICLP 2002. LNCS, vol. 2401, p. 255. Springer, Heidelberg (2002)

8. Buneman, P., Fernandez, M., Suciu, D.: UnQL: a query language and algebra for semistructured data based on structural recursion. The VLDB Journal 9(1), 76–110 (2000)
9. Cirstea, H., Kirchner, C.: The rewriting calculus - Parts I and II. Logic Journal of the IGPL 9(3) (2001)
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Science 285(2), 187–243 (2002)
11. Coelho, J., Florido, M.: CLP(Flex): Constraint logic programming applied to XML processing. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3291, pp. 1098–1112. Springer, Heidelberg (2004)
12. Coelho, J., Florido, M.: XCentric: A logic programming language for XML. Technical Report Dcc-2005-X, Dcc-Fc and Liacc, University of Porto (2005)
13. Dundua, B., Kutsia, T.: PρLog. Version 0.7, `http://www.risc.uni-linz.ac.at/people/tkutsia/software.html`
14. Frühwirth, T.: Theory and practice of Constraint Handling Rules. J. Logic Programming 37(1-3), 95–138 (1998)
15. Hosoya, H., Pierce, B.C.: XDuce: A statically typed XML processing language. ACM Trans. Internet Techn. 3(2), 117–148 (2003)
16. Kutsia, T.: Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols. PhD thesis, Johannes Kepler University, Linz (2002)
17. Kutsia, T., Marin, M.: Matching with regular constraints. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 215–229. Springer, Heidelberg (2005)
18. Maier, D.: Database desiderata for and XML query language (1998), `http://www.w3.org/TandS/QL/QL98/pp/maier.html`
19. Marin, M., Kutsia, T.: Foundations of the rule-based system RhoLog. Journal of Applied Non-Classical Logics 16(1-2), 151–168 (2006)
20. Martí-Oliet, N., Meseguer, J.: Rewriting logic: Roadmap and bibliography. Theoretical Computer Science 285(2), 121–154 (2002)
21. May, W.: XPath-Logic and XPathLog: a logic-programming-style XML data manipulation language. TPLP 4(3), 239–287 (2004)
22. The OBJ Family, `http://cseweb.ucsd.edu/~goguen/sys/obj.html`
23. REWERSE. Reasoning on the web, `http://rewerse.net/`
24. Schaffert, S.: Xcerpt: a rule-based query and transformation language for the Web. PhD thesis, University of Munich (2004)
25. van den Brand, M.G.J., van Deursen, A., Heering, J., de Jong, H.A., de Jonge, M., Kuipers, T., Klint, P., Moonen, L., Olivier, P.A., Scheerder, J., Vinju, J.J., Visser, E., Visser, J.: The ASF+SDF meta-environment: A component-based language development environment. In: Wilhelm, R. (ed.) CC 2001. LNCS, vol. 2027, pp. 365–370. Springer, Heidelberg (2001)
26. Visser, E.: Stratego: A language for program transformation based on rewriting strategies. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 357–362. Springer, Heidelberg (2001)

Learning to Rank Individuals in Description Logics Using Kernel Perceptrons[⋆]

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito

Dipartimento di Informatica, Università degli studi di Bari "Aldo Moro"
{fanizzi,claudia.damato,esposito}@di.uniba.it

Abstract. We describe a method for learning functions that can predict the ranking of resources in knowledge bases expressed in Description Logics. The method relies on a kernelized version of the PERCEPTRON RANKING algorithm which is suitable for batch but also online problems settings. The usage of specific kernel functions that encode the similarity between individuals in the context of knowledge bases allows the application of the method to ontologies in the standard representations for the Semantic Web. An extensive experimentation reported in this paper proves the effectiveness of the method at the task of ranking the answers to queries, expressed by class descriptions when applied to real ontologies describing simple and complex domains.

1 Introduction

Ranking a set of individual objects, as the result of relations sought between them and their relative relevance, is a fundamental task with a plenty of applications. Typically, ranked resources (e.g. documents, web services) may be returned as a result of *retrieval* process (from a corpus, a database, a directory, etc.). When the relevance of the outcomes depends exclusively on the query specification, this task is quite well-understood and many effective solutions exist, even for approximate cases. However, the problem likely turns out to be much harder when a general and precise measure of the relevance of the results is too complex or unavailable (e.g. multiple relevance orders, subjective user-dependent preferences, etc.).

Essentially, based on a request (a query) and, possibly, on some previous partial indications of an intended relevance (e.g. some feedback from the user), the set of retrieved resources must be ordered according to such indications. It may be possible to (partially) elicit the required information exploiting imprecise criteria that can often be expressed by means of examples rather than in a general and formal way. A related process of result ranking based on *relevance-feedback* mechanisms is represented by the task known as *collaborative filtering* which aims at detecting the relevance of information items for new users based on rankings previously acquired from others. All such problems can be cast in the framework of *inductive learning from examples* [9]. Given previously rated instances (e.g. movies, songs, etc.), the aim is to induce a hypothesis

[⋆] This work was partially funded by the MBLab FAR Project (MIUR DM19410). A short version of this paper has been accepted to ECAI2010 [6].

(e.g. learn a function) which can be then used to assign a meaningful ranking to new incoming instances of uncertain (unknown) ranking.

Even more difficult it is to adapt such problem settings and solutions to work with semantic knowledge bases. Their logic-oriented nature and inherent incompleteness requires *ad hoc* formalizations and suitable methods. On one hand, related works tackle the structure of the underlying relations (e.g. see [1, 7]) which cannot fully exploit the richness of the underlying logic representation. On the other hand, purely logical methodologies investigated in recent works may fall short in terms of scalability. For example, the related problem of semantic *matchmaking* is tackled by means of non-monotonic inferences in Description Logics (henceforth DLs) such as concept *abduction* and *contraction* [3]. Although elegant and well-founded, these approaches suffer from some drawbacks: the language-dependence of some of the required operations and the consequent complexity of the inference services with the growing expressiveness of the considered representation. Other frameworks tackle the problem by explicitly representing the preferences in the knowledge bases with conditional statements for ranking objects in ontologies [10]. Other approaches recur to fuzzy extensions of the standard DL languages to offer alternate ways for finding the best (top k) answers to queries [12].

In this work we tackle the problem of learning rank in DLs when relevance can be expressed only through instances labeled with their intended ranking. Although, as mentioned, this could be cast as a *classification* or *regression* problem (depending on the nature of the rankings), an off-the-shelf learning method adapted to DLs would not fully profit by the ordering relationships among the ranks. Non-parametric statistical learning has been shown to provide valid techniques to produce alternative models that enable forms of inductive reasoning in DLs [5]. Also the related task of *ranking* can be performed inductively, casting it as the problem of learning a suitable function from a training set of ranked instances. Specifically, we will resort to the *kernel perceptron* model [11] which exploits the relative ordering between instances and has also the advantage of being used in a *batch* or an *online* mode.. Ideally, even the number of ranks need not be specified, though typically the training data comes with a relative ordering, specified by the assignment to one of an ordered sequence of labels.

2 Ranking Problems

In general *learning to rank* can be described as follows. Given a set of ranked examples (x, y), where each individual object x in some input space $\mathcal{X}$ is assigned with a label y from an ordered set Y, the goal is to predict the rank for new unlabeled instances $(x, \cdot)$. This could be tackled as a *regression* or *classification* problem [9] by treating the ranks as real-values or the assignment to a particular rank value as a classification. Formally:

Definition 2.1 (ranking problem). *Let $\mathcal{X}$ be a set of individual objects and Y be a set of ranks, endowed with a total order relation $\leq_Y$, such that one of the following holds:*

- $|Y| = r < \infty$ *or* *(discrete case)*
- $\exists$ *bijection* $\beta : Y \mapsto \mathbb{R}^+$ *(continuous case)*

A ranking problem *is specified as follows:*
given: *a sample set* $S = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subseteq \mathcal{X} \times Y$,

where $y_i \in Y$ *is the* rank *of* $x_i \in \mathcal{X}$
find: *a* ranking function $\rho : \mathcal{X} \mapsto Y$ *inducing a relation* $\preceq_\rho$ *over* $\mathcal{X}$ *that is a partial (resp. total) order.*

Subscripts will be omitted when the context makes them obvious. The simplest way to define the set of ranks in the continuous case is $Y = \mathbb{R}^+$ or, in the other case, $Y = \{1, \ldots, r\}$.

In this setting x_i is *preferred* over x_j, denoted $x_i \preceq_\rho x_j$, iff $\rho(x_i) \leq_Y \rho(x_j)$. The objects x_i and x_j cannot be ordered (*not comparable*) when $y_i =_\rho y_j$, i.e. $\rho(x_i) \leq_Y \rho(x_j)$ and $\rho(x_j) \leq_Y \rho(x_i)$. In the categorical (discrete) case, one may consider that the (partial) ordering partitions the input space into r equivalence classes.

A reduction to a classification problem in the discrete case, would not make use of all of the available information; on the other hand the flexibility inherent in the ordering requirement is the price for the reduction to regression. on the other hand, in a regression setting a specific metric is required to convert the ranks into real values. This may be difficult in general and makes regression more sensitive to the rank representation rather than to their ordering [8]. It is therefore preferable to treat ranking as a problem in its own right and design specific algorithms able to take advantage of the specific nature of that problem [11]. Of course ranking functions are not bound to be based on linear components. This can be a by-product of the adoption of specific kernels and the related embedding.

The discrete case setting can be transposed into the problem of predicting the relative ordering of all possible pairs of examples, hence obtaining two-class classification problems. The drawback of this approach would be the extra computational effort required since the sample size for the algorithm grows quadratically with the number of examples. If, on the other hand, the training data is given in the form of all relative orderings, a set of ranks can generated as equivalence classes of the equality relation with the induced ordering.

3 Learning with DL-Kernels

A learning task requires finding an inductive model (i.e. a hypothesis function h) which can be adopted by a decision procedure to predict, given an input instance, the correct value for the function to be approximated efficiently (evaluating a simple model, such as a linear function) and effectively (close approximation). Kernel methods [9, 11] are particularly well suited from an engineering viewpoint because the learning algorithm (*inductive bias*) and the choices of the kernel function (*language bias*) are almost completely independent.

Discovering linear relations has a long tradition of research, with algorithms that are both efficient and well understood. A computational shortcut makes it possible to represent linear models efficiently in high-dimensional spaces to ensure adequate representational power: a kernel function. Different kernel functions may be related to different hypothesis spaces. Hence, the same kernel machine can be applied to different representations, provided that suitable kernel functions are available. Kernel methods are characterized by two aspects:

– data items are embedded (implicitly) into a vector space, the *feature* (*embedding*) space; this depends on the specific data types and domain knowledge expected for the particular data source;
– linear models are sought among the images of the data items in the feature space; the algorithms are implemented in such a way that only the inner products of the embedded points are needed and the products can be computed efficiently directly from the original data items using a *kernel* function.

The *learning* component is general-purpose, robust and also efficient, requiring an amount of computational resources that is polynomial in the size and number of data items even when the dimension of the embedding space grows exponentially. An algorithm may be adapted to structured spaces (e.g. trees, graphs [11]) by merely replacing the kernel function with a suitable one. Examples of the target concepts are to be provided to the learning algorithm to produce a definition for the target function in the form of a linear decision function depending on a tuple of weights.

Many learning problems can be reduced to the approximation of linear functions. In a simplified setting, consider an *input* space $\mathcal{X}$ of training instances represented by (binary or real) tuples ($\mathcal{X} = \{0,1\}^d$ or $\mathcal{X} = \mathbb{R}^d$) extended with an additional categorical feature $y \in Y$ (e.g. $Y = \{-1,+1\}$) indicating the membership w.r.t. an implicit target class: $(\boldsymbol{x}, y) \in \mathcal{X} \times Y$ (these are named *examples*). We will consider learning from a *sample* $S \subseteq (\mathcal{X} \times Y)$ of N examples.

A prototypical algorithm is the PERCEPTRON, a well-known procedure for learning the coefficients of linear classifiers [9]. For each incoming training example $(\boldsymbol{x}^i, y^i)$, the algorithm predicts a value according to the decision function determined by the current choice of weights $\boldsymbol{w}$. It can be written as depending on a dot product $\mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}) = \mathrm{sgn}(\sum_j w_j x_j \cdot \boldsymbol{x}) = \mathrm{sgn}(\sum_j \alpha_j (\boldsymbol{x}^i \cdot \boldsymbol{x}))$ which is a common characteristic of these methods. The algorithm compares the outcome with the correct label which is known for the examples considered during the training phase. On erroneous predictions, the weights $\boldsymbol{w}$ are revised depending on the set of examples that provoked a mistake: $\boldsymbol{w} = \boldsymbol{w} + \eta(y^i - \boldsymbol{w} \cdot \boldsymbol{x}^i)\boldsymbol{x}^i$.

The choice of kernel functions is very important as their computation should be efficient enough for controlling the complexity of the overall learning process. Although some kernels have been proposed for instances expressed in FOL fragments, only a few kernel functions for individuals have been proposed in the literature [5].

In this work, we resort to a set of general kernels for individuals in DL knowledge bases [5], that exploit a notion of similarity in the context of a set of concepts:

Definition 3.1 (DL kernel functions). *Given a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a set of concept descriptions $\mathsf{F} = \{F_1, F_2, \ldots, F_m\}$ defined in $\mathcal{T}$, the* kernel function *based on* F *and $p \in \mathbb{R}_+$, is a mapping $k_p^{\mathsf{F}} : \mathsf{Ind}(\mathcal{A}) \times \mathsf{Ind}(\mathcal{A}) \mapsto [0,1]$ defined: $\forall a, b \in \mathsf{Ind}(\mathcal{A})$ $k_p^{\mathsf{F}}(a,b) = \sqrt[p]{\sum_{i=1}^{m} (\kappa_i(a,b))^p}$, where the* simple kernel function *κ_i is defined $\forall i \in \{1, \ldots, m\}$:*

$$\kappa_i(a,b) = \begin{cases} 1 & \mathcal{K} \models F_i(a) \wedge \mathcal{K} \models F_i(b) \ \vee \ \mathcal{K} \models \neg F_i(a) \wedge \mathcal{K} \models \neg F_i(b) \\ 0 & \mathcal{K} \models \neg F_i(a) \wedge \mathcal{K} \models F_i(b) \ \vee \ \mathcal{K} \models F_i(a) \wedge \mathcal{K} \models \neg F_i(b) \\ u_i & otherwise \end{cases}$$

The rationale for these kernels is that similarity between individuals is determined by their similarity w.r.t. each concept in a given committee of features. Two individuals are maximally similar w.r.t. a given concept F_i if they exhibit the same behavior, i.e. both are instances of the concept or of its negation. Conversely, the minimal similarity holds when they belong to opposite concepts. Because of the OWA, sometimes a reasoner cannot ascertain the concept-membership of some individuals, hence, since both possibilities are open, an intermediate value u_i is assigned to reflect such uncertainty. This value (normally a .5 value is considered in case of total uncertainty) should estimate the probability that two instances may belong (not belong) to the extension of F_i based on the sizes of the *retrieval* sets of F_i and its complement w.r.t. the current ABox. In previous works the F_i's have been chosen randomly among the named concepts of the KB, although ad hoc methods for a choice of optimized sets of features have also been proposed [4]. The parameter p was borrowed from the form of the Minkowski's measures. Once the feature set is fixed, the possible values for the kernel function are determined, hence p has an impact on the granularity of the measure. As mentioned, instance-checking is to be employed for assessing the value of the simple similarity functions. Yet this is known to be computationally expensive (also depending on the specific DL language of choice). Alternatively, especially for ontologies that are rich of explicit class-membership information (assertions), a simple look-up may be sufficient, as suggested by the first definition of the κ_i functions.

4 Kernel-Based Ranking for DL Spaces

Preliminarily, we will assume an implicit kernel-defined feature space with the corresponding feature mapping ϕ so that $\phi(x_i)$ is in $\mathbb{R}^n$ for some n, $1 \leq n \leq \infty$.

A *linear ranking rule* function embeds the input data into the real axis by means of a linear function in the feature space $f(x) = (\boldsymbol{w} \cdot \phi(x))$. The real-value can be then converted to a rank by means of an r-dimensional vector of thresholds $\boldsymbol{\theta}$ with the θ_y's ordered according to the underlying relation on Y, i.e. $y \leq y'$ implies $\theta_y \leq \theta_{y'}$.

Definition 4.1 (ranking rule). *Given a kernel function* $\kappa : \mathcal{X} \mapsto \mathbb{R}$, *the ranking of an instance* x *is defined*[1] *by:* $\rho(x; \boldsymbol{w}, \boldsymbol{\theta}) = \min\{y \in Y \mid f(x) < \theta_y\}$, *or, in a dual representation, where* $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \phi(x_i)$*:*

$$\rho(x; \boldsymbol{\alpha}, \boldsymbol{\theta}) = \min \left\{ y \in Y \mid f(x) = \sum_{i=1}^{N} \alpha_i \kappa(x, x_i) < \theta_y \right\}$$

Such a ranking rule partitions $\mathcal{X}$ into $r+1$ equivalence classes corresponding to parallel bands orthogonal w.r.t. $\boldsymbol{w}$ and delimited by the thresholds θ_i (to be learned). Depending on the kernel function adopted κ one may obtain linear or even non linear intervals. Note that the classes need not be equally spaced. Moreover, the objects within each class can further be ordered also by the value of the function $f(x)$.

In order to learn the parameters, we devised DL-KPRank, a kernelized version of the Perceptron Ranking algorithm PRank [2]. The DL-KPRank algorithm is defined by rounds (iterations) of the KPRankUpdate function shown as Algorithm 1.

[1] It is assumed that the largest label r is assigned with a large θ_r so that the minimum exists.

Algorithm 1. The core updating function of the DL-KPRank algorithm.

```
function KPRankUpdate($x_t$, $y_t$, $\boldsymbol{\alpha}^t$, $\boldsymbol{\theta}^t$): ($\boldsymbol{\alpha}^{t+1}$, $\boldsymbol{\theta}^{t+1}$)
$(x_t, y_t)$: ranked training example at round $t$
$\boldsymbol{\alpha}^t$, $\boldsymbol{\alpha}^{t+1}$: weights at rounds $t, t+1$
$\boldsymbol{\theta}^t$, $\boldsymbol{\theta}^{t+1}$: thresholds at rounds $t, t+1$
 1: $\boldsymbol{u}^t \leftarrow \mathbf{0}$, $\boldsymbol{\ell}^t \leftarrow \mathbf{0}$                                   {initialization}
 2: $\hat{y} \leftarrow \rho(x_t; \boldsymbol{\alpha}^t, \boldsymbol{\theta}^t)$                                   {prediction}
 3: if $\hat{y} \neq y_t$ then
 4:     for $j = 1$ to $r-1$ do
 5:         if $j \geq y_t$ then $\ell_j^t \leftarrow -1$ else $\ell_j^t \leftarrow +1$ endif
 6:     end for
 7:     for $j = 1$ to $r-1$ do
 8:         if $(\sum_{i=1}^{N} \alpha_i^t \kappa(x_t, x_i) - \theta_j^t)\ell_j^t \leq 0$ then $u_j^t \leftarrow \ell_j^t$ endif
 9:     end for
10:     $\boldsymbol{\alpha}^{t+1} \leftarrow \boldsymbol{\alpha}^t + y_t \sum_{j=1}^{r-1} u_j^t$                     {weights update}
11:     $\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^t - \boldsymbol{u}^t$                              {thresholds update}
12: else
13:     $\boldsymbol{\alpha}^{t+1} \leftarrow \boldsymbol{\alpha}^t$
14:     $\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^t$
15: end if
16: return ($\boldsymbol{\alpha}^{t+1}$, $\boldsymbol{\theta}^{t+1}$)
```

The objective of the algorithm is to find a Perceptron weight vector $\boldsymbol{\alpha}$ which successfully projects all the instances in $\mathcal{X}$ into the r subintervals defined by the thresholds $\boldsymbol{\theta}$, i.e. for the rank of j the subinterval is $\theta_{j-1} < f(x) < \theta_j$.

At round t the first step is to predict the rank $\hat{y}_t$ (line 2) for a given instance x_t by selecting the smallest rank y such that $f(x) < \theta_y$. If the prediction $\hat{y}_t$ is not the correct rank then a label of $\ell_j^t = +1$ is allocated to those subintervals above the target rank y_t and $\ell_j^t = -1$ to those below (lines 3, 4 and 5). For each subinterval, if the ranking rule based on $\boldsymbol{\alpha}^t$ and $\boldsymbol{\theta}^t$ misclassifies the label ℓ_j^t then the label is subtracted from the threshold θ_j^t, and the Perceptron weights vector is updated. The rationale is that updating $\boldsymbol{\alpha}^t$ and $\boldsymbol{\theta}^t$ in this way has the effect of moving the threshold of the desired rank θ_j^{t+1} and the updated predicted rank $f(x_t)$ closer together. This procedure is repeated for all the subintervals $j = 1, \ldots, r-1$ for round t.

The call to the update routine is inserted in an outer loop which repeats the weight and threshold optimization until a satisfactory rate of training examples is correctly ranked by the resulting function (low or null average loss).

In order to use the ranking algorithm with objects described in a DL knowledge base $\mathcal{K}$, let us will consider a training set of $TSet \subseteq \mathsf{Ind}(\mathcal{A})$. Then the related Gram matrix $\boldsymbol{K}$ of the kernel values w.r.t. the couples of individuals can be obtained by choosing κ as one of the kernels in the family defined in Def. 3.1, $\kappa = k_p^{\mathsf{F}}$, based on a context F of concepts (for example, the leaf concepts of the subsumption hierarchy in $\mathcal{T}$) and a choice of p (e.g. 2).

Table 1. Facts concerning the ontologies employed in the experiments

ontology	*DL language*	*#concepts*	*#object props.*	*#data props.*	*#individuals*
MDM0.37	$\mathcal{ALCROF}(\mathbf{D})$	196	22	3	103
WINE	$\mathcal{SHOIN}(\mathbf{D})$	140	21	1	206
BIOPAX	$\mathcal{ALCHF}(\mathbf{D})$	28	19	30	323
LUBM	$\mathcal{ALR}^{+}\mathcal{HI}(\mathbf{D})$	43	7	25	555
HD	$\mathcal{ALCIN}(\mathbf{D})$	1449	10	10	639
NTN	$\mathcal{SHIF}(\mathbf{D})$	47	27	8	676
SWSD	$\mathcal{ALCH}$	258	25	0	732
FINANCIAL	$\mathcal{ALCIF}$	60	17	0	1000

5 Experimental Evaluation

The DL-RANK system implements the training method and ranking procedure explained in the previous sections, borrowing the square kernels of the family (i.e. those with $p = 2$). Its performance has been tested in a large number of ranking problems with ontological knowledge bases. In the following, we present the experimental setting and discuss the outcomes.

A number of OWL ontologies describing different domains have been selected, namely: MDM0.37, WINE, NEWTESTAMENTNAMES (NTN) from the Protégé library[2], the *Semantic Web Service Discovery* dataset[3] (SWSD), an ontology generated by the Lehigh University Benchmark[4] (LUBM), the BioPax glycolysis ontology[5](BioPax), HUMANDISEASE[6] (HD) that has been developed for the *Open Biomedical Ontologies* project[7], and the FINANCIAL ontology[8]. Tab. 1 summarizes salient details regarding these ontologies. For each ontology, 100 satisfiable query concepts were randomly generated by composition (conjunction and/or disjunction) of 2 through 8 concepts: named concepts and also (universal and existential) role restrictions. Since, differently from other ranking contexts, no specific DL dataset with a known baseline to be compared is available, the reference ranking y_t for each individual x_t was derived by recurring to an approximated classification procedure based on the k-*nearest neighbor* principle [9] that returned the likelihood measure: $\Pr(h_Q(x_t) = +1 \mid x_t) = \pi_{+1}\hat{D}_{+1}(x_t)/\sum_v \pi_v \hat{D}_v(x_t)$, where h_Q is the induced classification function, π's are priors of getting the respective classification and the $\hat{D}$'s are estimates of the density function, for the given classification value, around the input individual. The likelihood of the class-membership mapped each individual to the uniformly-sized bin of the unit interval, corresponding to one of the possible rank values ($r = |Y|$ was set to 5).

[2] http://protege.stanford.edu/plugins/owl/owl-library
[3] https://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Projects/xmedia/dl-tree.htm
[4] http://swat.cse.lehigh.edu/projects/lubm
[5] http://www.biopax.org/Downloads/Level1v1.4/biopax-example-ecocyc-glycolysis.owl
[6] http://www.jalojavier.es/humandisease.owl
[7] http://www.obofoundry.org
[8] http://www.cs.put.poznan.pl/alawrynowicz/financial.owl

Table 2. Experimental results: average rank loss values ± average standard deviations

ontology	*avg. loss* ± *std. deviation*	*ontology*	*avg. loss* ± *std. deviation*
MDM0.37	0.06 ± 0.03	WINE	0.25 ± 0.03
BIOPAX	0.27 ± 0.01	LUBM	0.29 ± 0.04
NTN	0.13 ± 0.04	HD	0.03 ± 0.02
SWSD	0.13 ± 0.08	FINANCIAL	0.16 ± 0.03

Experimentally, it was observed that large training sets make the kernel functions more accurate. The squared kernels (k_2^{F}) was employed from the family, using all the named concepts in the knowledge base for determining the committee of features F with no further optimization. A standard OWL reasoner (PELLET v. 2.0.1) was employed for computing the instance checks that are required by the kernel functions.

In order to determine a good estimate of the accuracy of the inductive ranking, reducing the variance due to the composition of the specific training/test sets during the various runs, the experiments have been replicated selecting training and test sets according to the standard *.632+ bootstrap* procedure [9]. This procedure requires creating random sets of training examples a repeated sampling (with replacement) from the population of examples; this produces sets which roughly amount to 63.2% of the total set of examples, while the complement sets are used as respective tests sets. The resulting measure, which may give a too optimistic estimate of the real performance, is adjusted to take into account both the training and the test errors properly weighted. The adopted performance index was an average measure of *rank loss* [2, 8], simply defined as: $\text{avg\_loss}(\mathsf{TestSet}) = \frac{1}{T}\sum_{i=1}^{T} |\hat{y}_i - y_i|/r$, where the y_i and $\hat{y}_i$ stand, resp., for the correct rank and the predicted one for test instances (and $T = |\mathsf{TestSet}|$).

Tab. 2 reports the outcomes in terms of the average rank loss and its variance. These outcomes show that the method is sufficiently accurate even for the cases of small ontologies (where the size is intended in the number of individuals). Indeed, as with other inductive methods the performance of the produced model tends to improve with the availability of more training instances. A noteworthy exception is represented by the experiment with the MDM0.37 ontology which produced a good ranker with small training sets as testified by the extremely limited average loss (6%).

As regards the largest ontologies (NTN, HD, SWSD, and FINANCIAL) the results show that the ranking functions produced were, on average, more accurate (average losses below 20%) and, in one case (HD) below 10% (namely 3%). The large number of queries (100) used in the experiments on the various ontologies and the random selection of the training sets in each repetition of the experiments guarantees the significance of these outcomes. This is also testified by the low standard deviation of the loss in all of the performed experiments. An exception is represented by the case of the SWSD ontology, for which the highest variability was found. This can be explained by the particular sparseness of the population of this ontology, which was originally designed for the purpose of semantic Web service discovery (transposing another standard ontology in to OWL). Indeed, in spite of the many classes therein (see Tab. 1), very few individuals are available as instance of each class, that makes the similarity values computed by the kernel function a less significant.

6 Conclusions and Future Work

We have presented some solutions to the problem of ranking individuals in DL knowledge bases. Since it is difficult to define a ranking of (retrieved) resources can be hardly with a general (logical) encoding of preferences performed, a novel statistical method for learning ranking functions from examples was introduced. The advantages of method based on kernels is that complex (non-linear) ordinal relations can be discovered in the space of individuals, while working on linear models (in the embedding space) with the consequent efficiency. Even more so, the method is suitable for an online utilization, improving its performance as new ranked instances are available. Since no testbeds for these specific representation seem to exist so far, we crafted a method for producing training sets of ranked individuals w.r.t. query concepts expressed in DLs, which allowed an extensive experimentation with a number of various real ontologies.

Extensions will concern the investigation of alternative algorithms derived from support vector regression and the enhancement of the presented one by averaging the models obtained in the various update loops according to standard methods, such as *bagging* or *Bayes point* estimation [9]. Moreover, new datasets shall be derived from the standard testbeds employed for related tasks (e.g. collaborative filtering).

References

[1] Aleman-Meza, B., Halaschek-Wiener, C., Arpinar, I., Ramakrishnan, C., Sheth, A.P.: Ranking complex relationships on the semantic web. IEEE Internet Computing 9(3), 37–44 (2005)

[2] Crammer, K., Singer, Y.: Pranking with ranking. In: Dietterich, T., et al. (eds.) Advances in Neural Information Processing Systems, vol. 14, pp. 641–647. MIT Press, Cambridge (2001)

[3] Di Noia, T., Di Sciascio, E., Donini, F.M.: Semantic matchmaking as non-monotonic reasoning: A description logic approach. J. Artif. Intell. Res. 29, 269–307 (2007)

[4] Fanizzi, N., d'Amato, C., Esposito, F.: Evolutionary conceptual clustering based on induced pseudo-metrics. Semantic Web Information Systems 4(3), 44–67 (2008)

[5] Fanizzi, N., d'Amato, C., Esposito, F.: Statistical learning for inductive query answering on OWL ontologies. In: Sheth, A., et al. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 195–212. Springer, Heidelberg (2008)

[6] Fanizzi, N., d'Amato, C., Esposito, F.: Towards learning to rank in description logics. In: Proc. of the European Conference on Artificial Intelligence, ECAI 2010 (to appear, 2010)

[7] Franz, T., Schultz, A., Sizov, S., Staab, S.: Triplerank: Ranking semantic web data by tensor decomposition. In: Bernstein, A., et al. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 213–228. Springer, Heidelberg (2009)

[8] Harrington, E.F.: Online ranking/collaborative filtering using the perceptron algorithm. In: Fawcett, T., Mishra, N. (eds.) Proc. of ICML 2003, pp. 250–257. AAAI Press, Menlo Park (2003)

[9] Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. In: Data Mining, Inference, and Prediction. Springer, Heidelberg (2001)

[10] Lukasiewicz, T., Schellhase, J.: Variable-strength conditional preferences for ranking objects in ontologies. Web Semant. 5(3), 180–194 (2007)

[11] Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)

[12] Straccia, U.: Towards top-k query answering in description logics: The case of DL-Lite. In: Fisher, M., et al. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 439–451. Springer, Heidelberg (2006)

A Probabilistic Abduction Engine for Media Interpretation Based on Ontologies

Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel

Hamburg University of Technology, Germany

Abstract. We propose an abduction-based formalism that uses description logics for the ontology and Horn rules for defining the space of hypotheses for explanations, and we use Markov logic to define the motivation for the agent to *generate* explanations on the one hand, and for *ranking* different explanations on the other. The formalism is applied to media interpretation problems in a agent-oriented scenario.[1]

1 Introduction

For multimedia interpretation in the context of an agent-based scenario, and for the combined interpretation of information coming from different modalities in particular, a semantically well-founded formalization is required. Low-level percepts, which are represented symbolically, define the observations of an agent w.r.t. some content, and interpretations of the content are defined as explanations for the observations. In [1] we have proposed an abduction-based formalism that uses description logics for the ontology and Horn rules for defining the space of hypotheses for explanations (i.e., the space of possible interpretations of media content). An evaluation of the abduction approach based on description logics and rules is presented in [3]. A discussion of related work can be found in [4].

In this paper, we propose the use of a probabilistic logic to define the motivation for the agent to *generate* explanations on the one hand, and for *ranking* different explanations on the other. Furthermore, we discuss how the interpretation process is performed, possibly with uncertainty and inconsistency in the input data. We also introduce a new approach for ranking interpretation Aboxes. The explanationranking process is performed based on a probabilistic scoring function (as opposed to the proof-theoretic scoring function used in [3]). A termination condition is also defined which determines how long the interpretation process should be performed. The approach is evaluation using a detailed example.

Due to space restrictions, not all preliminaries could be specified in this paper. For an introduction to description logics, grounded conjunctive queries, and rules we refer to [3]. For specifying the ontology used to describe low-level analysis

[1] This work has been funded by the European Community with the project CASAM (Contract FP7-217061 CASAM) and by the German Science Foundation with the project PRESINT (DFG MO 801/1-1).

results as well as high-level interpretation results, a less expressive description logic is applied to facilitate fast computations. We decided to represent the domain knowledge with the DL $\mathcal{ALH}_f{}^-$ (restricted attributive concept language with role hierarchies, functional roles and concrete domains). The motivation to only allow a restricted use of existential restrictions is to support a well-founded integration of the description logic part of the knowledge base with the probabilistic part, based on Markov logics.

The Markov logic formalism [2] provides a means to combine the expressivity of first-order logic augmented with the formalism of Markov networks [6]. The Markov logic formalism uses first-order logic to define "templates" for constructing Markov networks. The basic notion for this is called a Markov logic network [2].

A Markov logic network $MLN = (\mathcal{F}_{MLN}, \mathcal{W}_{MLN})$ consists of a sequence of first-order formulas $\mathcal{F}_{MLN} = \langle F_1, ..., F_m \rangle$ and a sequence of real number weights $\mathcal{W}_{MLN} = \langle w_1, ..., w_m \rangle$. The association of a formula to its weight is by position in the sequence. For a formula $F \in \mathcal{F}_{MLN}$ with associated weight $w \in \mathcal{W}_{MLN}$ we also write $w\, F$ (weighted formula). Weights are used to specify probability distributions. For a more detailed introduction to description logics and their combination with Markov logic networks, we refer to [5].

The central idea is to use abduction to compute possible explanations for observations of an agent, which are seen as high-level interpretations. The space of abducibles is defined in terms of Horn rules in combination with ontologies (see [3] for details). Compared to other approaches (e.g., [7]) also the ontology is used for checking whether something must be abduced. In addition, a motivation for computing explanations (or interpretations) using abduction is given by assuming that the agent would like to increase increase the probability that the observations are true. If there is no significant increase (due to a threshold ϵ), possible interpretations are considered as irrelevant for the agent.[2] Another important idea is that, given a "current" interpretation, the agent should be able to compute what must be added due to new percepts and what must be retracted (for this purpose, an Abox difference operator is used).

The abduction and interpretation procedures are discussed in detail in Section 2. In Section 3, a complete example is given showing the main approach using intermediate steps. Section 4 summarizes this paper.

2 Probabilistic Interpretation Engine

At the beginning of this section, the most important preliminaries to the abduction process are specified. Afterwards, functions are introduced for the abduction procedure, interpretation procedure, and the media interpretation agent.

2.1 Computing Explanations

In general, abduction is formalized as $\Sigma \cup \Delta \models_{\mathcal{R}} \Gamma$ where background knowledge (Σ), rules ($\mathcal{R}$), and observations (Γ) are given, and explanations (Δ) are

[2] Obviously, there is a horizon problem, which we neglect for the time being.

to be computed. In terms of DLs, Δ and Γ are Aboxes and Σ is a pair of Tbox and Abox. Abox abduction is implemented as a non-standard retrieval inference service in DLs. In contrast to standard retrieval inference services where answers are found by exploiting the ontology, Abox abduction has the task of acquiring what should be added to the knowledge base in order to answer a query. Therefore, the result of Abox abduction is a set of hypothesized Abox assertions. To achieve this, the space of abducibles has to be defined. We do this in terms of rules. We assume that a set of rules $\mathcal{R}$ as defined in [3] are specified, and use a function *explanation_step*, see [3] or [5] for details.

2.2 The Abduction Procedure

In the following, we devise an abstract computational engine for "explaining" Abox assertions in terms of a given set of rules. Explanation of Abox assertions w.r.t. a set of rules is meant in the sense that using the rules some high-level explanations are constructed such that the Abox assertions are entailed. The explanation of an Abox is again an Abox. For instance, the output Abox represents results of the content interpretation process. Let the agenda $\mathfrak{A}$ be a set of Aboxes Γ and let Γ be an Abox of observations whose assertions are to be explained. The goal of the explanation process is to use a set of rules $\mathcal{R}$ to derive "explanations" for elements in Γ. The explanation algorithm implemented in the Conceptual Abduction Engine (CAE) works on a set of Aboxes $\mathfrak{I}$.

The complete explanation process is implemented by the CAE function:

```
**Function** CAE($\Omega$, $\Xi$, $\Sigma$, $\mathcal{R}$, $S$, $\mathfrak{A}$):
**Input:** a strategy function $\Omega$, a termination function $\Xi$, a knowledge
base $\Sigma$, a set of rules $\mathcal{R}$, a scoring function $S$, and an agenda $\mathfrak{A}$
**Output:** a set of interpretation Aboxes $\mathfrak{I}'$
$\mathfrak{I}' := \{assign\_level(l, \mathfrak{A})\}$;
**repeat**
    $\mathfrak{I} := \mathfrak{I}'$;
    $(\mathcal{A}, \alpha) := \Omega(\mathfrak{I})$;
    $l = l + 1$;
    $\mathfrak{I}' := (\mathfrak{A} \setminus \{\mathcal{A}\}) \cup assign\_level(l, explanation\_step(\Sigma, \mathcal{R}, S, \mathcal{A}, \alpha))$;
**until** $\Xi(\mathfrak{I})$ *or no* $\mathcal{A}$ *and* $\alpha$ *can be selected such that* $\mathfrak{I}' \neq \mathfrak{I}$ ;
**return** $\mathfrak{I}'$
```

where $assign\_level(l, \mathfrak{A})$ is defined as follows:

$$assign\_level(l, \mathfrak{A}) = map(\lambda(\mathcal{A}) \bullet assign\_level(l, \mathcal{A}), \mathfrak{A}) \tag{1}$$

$assign\_level(l, \mathfrak{A})$ takes as input a superscript l and an agenda $\mathfrak{A}$. In the following, $assign\_level(l, \mathcal{A})$ is defined which superscripts each assertion α of the Abox $\mathcal{A}$ with l if the assertion α does not already have a superscript:

$$assign\_level(l, \mathcal{A}) = \left\{\alpha^l \mid \alpha \in \mathcal{A}, \alpha \neq \beta^i, i \in \mathbb{N}\right\} \tag{2}$$

The motivation for adding levels to assertions is to support different strategies Ω. Note that l is a global variable, its starting value is zero, and it is incremented in the CAE function. The *map*[3] function is defined as follows:

$$map(f, X) = \bigcup_{x \in X} \{f(x)\} \tag{3}$$

It takes as parameters a function f and a set X and returns a set consisting of the values of f applied to every element x of X. The CAE function applies the strategy function Ω in order to decide which assertions to explain, uses a termination function Ξ in order to check whether to terminate due to resource constraints and a scoring function S to valuate an explanation. The function Ω for the explanation strategy and Ξ for the termination condition are used as an oracle and must be defined in an application-specific way.

In the next Section we explain how probabilistic knowledge is used to (i) formalize the effect of the "explanation", and (ii) formalize the scoring function S used in the CAE algorithm explained above. In addition, it is shown how the termination condition (represented with the parameter Ξ in the above procedure) can be defined based on the probabilistic conditions.

2.3 The Interpretation Procedure

The interpretation procedure is completely discussed in this section by explaining the interpretation problem and presenting a solution to this problem. The solution is presented by a probabilistic interpretation algorithm which calls the CAE function described in the previous section. In the given algorithm, a termination function, and a scoring function are defined. The termination function determines if the interpretation process can be stopped since at some point during the interpretation process it makes no sense to continue the process. The reason for stopping the interpretation process is that no significant changes can be seen in the results. The defined scoring function in this section assigns probabilistic scores to the interpretation Aboxes.

Problem. The objective of the interpretation component is the generation of interpretations for the observations. An interpretation is an Abox which contains high level concept assertions. Since in this paper we adopt the view that agents are used for solving the problems while acquiring information, in the following the same problem is formalized in the perspective of an agent: Consider an agent given some percepts in an environment where the percepts are the analysis results of the multimedia documents.[4] The objective of this agent is finding explanations for the existence of percepts. The question is how the interpretation Aboxes are

[3] Please note that in this report, the expression *map* is used in two different contexts. The first one MAP denotes the Maximum A Posteriori approach which is a sampling method whereas the second one *map* is a function used in the $assign\_level(l, \mathfrak{A})$ function.

[4] The analysis might also be carried out by the agent.

determined and how long the interpretation process must be performed by the agent. The functionality of this Media Interpretation Agent is presented in the *MI_Agent* algorithm in Section 2.4.

Solution. In the following, an application for a probabilistic interpretation algorithm is presented which gives a solution to the mentioned problem. This solution illustrates a new perspective to the interpretation process and the reason why it is performed. In this approach, we define a probabilistic scoring function which assigns probabilities to the interpretation Aboxes. Additionally, we define a termination function which determines whether the interpretation process can be terminated. The central idea is to check whether interpretation results computed by a call to CAE substantially increase the probability the the observations are true. If there is no significant increase (due to a threshold ϵ, possible interpretations are considered as irrelevant for the agent.[5] Another important idea is that, given a "current" interpretation, the agent should be able to compute what must be added due to new percepts and what must be retracted (for this purpose, an Abox difference operator is used).

We are now ready to define the algorithm. Assume that the media interpretation component receives a weighted Abox $\mathcal{A}$ which contains observations. In the following, the applied operation $P(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T})$ in the algorithm is explained:

The $P(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T})$ function determines the probability of the Abox $\mathcal{A}$ with respect to the Abox $\mathcal{A}'$, a set of rules $\mathcal{R}$, a set of weighted rules $\mathcal{WR}$, and the Tbox $\mathcal{T}$ where $\mathcal{A} \subseteq \mathcal{A}'$. Note that $\mathcal{R}$ is a set of forward and backward chaining rules. The probability determination is performed based on the Markov logic formalism as follows:

$$P(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T}) = P_{MLN(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T})}(\vec{Q}(\mathcal{A}) \mid \vec{e}(\mathcal{A}')) \tag{4}$$

$\vec{Q}(\mathcal{A})$ denotes an event composed of the conjunction of all assertions which appear in the Abox $\mathcal{A}$. Assume that the Abox $\mathcal{A}$ contains n assertions $\alpha_1, \ldots, \alpha_n$. Consequently, the event for the Abox $\mathcal{A}$ is defined as follows:

$$\vec{Q}(\mathcal{A}) = \langle \alpha_1 = true \wedge \ldots \wedge \alpha_n = true \rangle \tag{5}$$

Consider Abox $\mathcal{A}$ contains m assertions $\alpha_1, \ldots, \alpha_m$. Then, the evidence vector $\vec{e}(\mathcal{A})$ is defined by:

$$\vec{e}(\mathcal{A}) = \langle \alpha_1 = true, \ldots, \alpha_m = true \rangle \tag{6}$$

Note that $\alpha_1, \ldots, \alpha_n$ denote the boolean random variables of the MLN. In order to answer the query $P_{MLN(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T})}(\vec{Q}(\mathcal{A}) \mid \vec{e}(\mathcal{A}'))$ the function MLN $(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T})$ is called. This function returns a Markov logic network $MLN = (\mathcal{F}_{MLN}, \mathcal{W}_{MLN})$ where $\mathcal{F}_{MLN}$ and $\mathcal{W}_{MLN}$ are ordered sets initialized as follows:

[5] Obviously, there is a horizon problem, which we neglect for the time being.

$\mathcal{F}_{MLN} = \emptyset$ and $\mathcal{W}_{MLN} = \emptyset$. In the following, it is described how the MLN is built based on the Aboxes $\mathcal{A}$ and $\mathcal{A}'$, the rules $\mathcal{R}$ and $\mathcal{WR}$ and the Tbox $\mathcal{T}$:[6]

$$MLN(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T}) = \begin{cases} \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{\alpha\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{w\} & \text{if } w\,\alpha \in \mathcal{A} \\ \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{\alpha\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{\infty\} & \text{if } \alpha \in \mathcal{A} \\ \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{\alpha\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{w\} & \text{if } w\,\alpha \in \mathcal{A}' \\ \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{\alpha\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{\infty\} & \text{if } \alpha \in \mathcal{A}' \\ \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{\alpha\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{\infty\} & \text{if } \alpha \in \mathcal{R} \\ \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{\alpha\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{w\} & \text{if } w\,\alpha \in \mathcal{WR} \\ \mathcal{F}_{MLN} = \mathcal{F}_{MLN} \cup \{FOL(\alpha)\};\; \mathcal{W}_{MLN} = \mathcal{W}_{MLN} \cup \{\infty\} & \text{if } \alpha \in \mathcal{T} \end{cases}$$

where w and α denote a weight and an assertion, respectively. In the following, the interpretation algorithm *Interpret* is presented:

```
**Function** Interpret($\mathfrak{A}$, $CurrentI$, $\Gamma$, $\mathcal{T}$, $\mathcal{FR}$, $\mathcal{BR}$, $\mathcal{WR}$, $\epsilon$)
**Input:** an agenda $\mathfrak{A}$, a current interpretation Abox $CurrentI$, an Abox of observations $\Gamma$, a Tbox $\mathcal{T}$, a set of forward chaining rules $\mathcal{FR}$, a set of backward chaining rules $\mathcal{BR}$, a set of weighted rules $\mathcal{WR}$, and the desired explanation significance threshold $\epsilon$
**Output:** an agenda $\mathfrak{A}'$, a new interpretation Abox $NewI$, and Abox differences for additions $\Delta_1$ and omissions $\Delta_2$
$i := 0$ ;
$\mathcal{R} := \mathcal{FR} \cup \mathcal{BR}$;
$p_0 := P(\Gamma, \Gamma, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ ;
$\Xi := \lambda(\mathfrak{A}) \bullet$
$\{i := i + 1; p_i := \max_{\mathcal{A} \in \mathfrak{A}} P(\Gamma, \mathcal{A} \cup \mathcal{A}_0, \mathcal{R}, \mathcal{WR}, \mathcal{T});$ **return** $\mid p_i - p_{i-1} \mid < \frac{\epsilon}{i}\}$;
$\Sigma := (\mathcal{T}, \emptyset)$;
$S := \lambda((\mathcal{T}, \mathcal{A}_0)), \mathcal{R}, \mathcal{A}, \Delta) \bullet P(\Gamma, \mathcal{A} \cup \mathcal{A}_0 \cup \Delta, \mathcal{R}, \mathcal{WR}, \mathcal{T})$;
$\mathfrak{A}' := CAE(\Omega, \Xi, \Sigma, \mathcal{R}, S, \mathfrak{A})$;
$NewI = argmax_{\mathcal{A} \in \mathfrak{A}'}(P(\Gamma, \mathcal{A}, \mathcal{R}, \mathcal{WR}, \mathcal{T}))$;
$\Delta^+ = AboxDiff(NewI, CurrentI)$; // additions
$\Delta^- = AboxDiff(CurrentI, NewI)$; // omissions
**return** $(\mathfrak{A}', NewI, \Delta^+, \Delta^-)$;
```

In the above algorithm, the termination function Ξ and the scoring function S are defined by lambda calculus terms. The termination condition Ξ of the algorithm is that no significant changes can be seen in the successive probabilities p_i and p_{i-1} (scores) of the two successive generated interpretation Aboxes in two successive levels $i-1$ and i. In this case, the current interpretation Abox $CurrentI$ is preferred to the new interpretation Abox $NewI$. The CAE function is called which returns the agenda $\mathfrak{A}'$. Afterwards, the interpretation Abox $NewI$ with the maximum score among the Aboxes $\mathcal{A}$ of $\mathfrak{A}'$ is selected. Additionally, the Abox differences Δ^+ and Δ^-, respectively, for additions and omissions among the interpretation Aboxes $CurrentI$ and $NewI$ are computed. In this paper, we formalize AboxDiff as set difference, knowing that a semantic approach would be desirable (see [5] for a semantics-based definition of *AboxDiff*).

[6] $FOL(\phi)$ represents the GCI ϕ in first-order notation.

In the following, the strategy condition Ω is defined which is one of the parameters of the CAE function:

```
**Function** $\Omega(\mathfrak{I})$
**Input:** a set of interpretation Aboxes $\mathfrak{I}$
**Output:** an Abox $\mathcal{A}$ and a fiat assertion $\alpha$
$\mathfrak{A} := \left\{\mathcal{A} \in \mathfrak{I} \mid \neg\exists \mathcal{A}' \in \mathfrak{I}, \mathcal{A}' \neq \mathcal{A} : \exists {\alpha'}^{l'} \in \mathcal{A}' : \forall \alpha^{l} \in \mathcal{A} : l' < l\right\}$;
$\mathcal{A} := random\_select(\mathfrak{A})$;
$min\_\alpha s = \left\{\alpha^{l} \in \mathcal{A} \mid \neg\exists {\alpha'}^{l'} \in \mathcal{A}', {\alpha'}^{l'} \neq \alpha^{l}, l' < l\right\}$;
**return** $(\mathcal{A}, random\_select(min\_\alpha s))$;
```

In the above strategy function Ω, the agenda $\mathfrak{A}$ is a set of Aboxes $\mathcal{A}$ such that the assigned superscripts to their assertions are minimum. In the next step, an Abox $\mathcal{A}$ from $\mathfrak{A}$ is randomly selected. Afterwards, the min_α_s set is determined which contains the assertions α from $\mathcal{A}$ whose superscripts are minimal. These are the assertions which require explanations. The strategy function returns as output an Abox $\mathcal{A}$ and an assertion α which requires explanation.

2.4 The Media Interpretation Agent

In the following, the *MI_Agent* function is presented which calls the *Interpret* function:

```
**Function** MI_Agent($Q$, $Partners$, $Die$, $(\mathcal{T}, \mathcal{A}_0), \mathcal{FR}, \mathcal{BR}, \mathcal{WR}, \epsilon$)
**Input:** a queue of percept results $Q$, a set of partners $Partners$, a function $Die$, a background knowledge base $(\mathcal{T}, \mathcal{A}_0)$, a set of forward chaining rules $\mathcal{FR}$, a set of backward chaining rules $\mathcal{BR}$, a set of weighted rules $\mathcal{WR}$, and the desired precision of the results $\epsilon$
**Output:** –
$CurrentI = \emptyset$;
$\mathfrak{A}'' = \{\emptyset\}$;
**repeat**
    $\Gamma := extractObservations(Q)$;
    $W := MAP(\Gamma, \mathcal{WR}, \mathcal{T})$ ;
    $\Gamma' := select(W, \Gamma)$;
    $\mathfrak{A}' := filter(\lambda(\mathcal{A}) \bullet consistent_{\Sigma}(\mathcal{A}),$
                $map(\lambda(\mathcal{A}) \bullet \Gamma' \cup \mathcal{A} \cup \mathcal{A}_0 \cup forward\_chain(\Sigma, \mathcal{FR}, \Gamma' \cup \mathcal{A} \cup \mathcal{A}_0),$
                    $\{select(MAP(\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0, \mathcal{WR}, \mathcal{T}), \Gamma' \cup \mathcal{A} \cup \mathcal{A}_0) \mid$
    $\mathcal{A} \in \mathfrak{A}''\}))$;
    $(\mathfrak{A}'', NewI, \Delta^{+}, \Delta^{-}) :=$
    $Interpret(\mathfrak{A}', CurrentI, \Gamma', \mathcal{T}, \mathcal{FR}, \mathcal{BR}, \mathcal{WR} \cup \Gamma, \epsilon)$;
    $CurrentI := NewI$;
    $Communicate(\Delta^{+}, \Delta^{-}, Partners)$;
    $\mathfrak{A}'' := manage\_agenda(\mathfrak{A}'')$;
**until** *Die()* ;
```

The body of *MI_Agent* uses a set of auxiliary functions, which are defined as follows.

$$\mathit{filter}(f, X) = \bigcup_{x \in X} \begin{cases} \{x\} & \text{if } f(x) = \mathit{true} \\ \emptyset & \text{else} \end{cases} \tag{7}$$

The function *filter* takes as parameters a function f and a set X and returns a set consisting of the values of f applied to every element x of X. In the *MI_Agent* function, the current interpretation *CurrentI* is initialized to the empty set and the agenda $\mathfrak{A}''$ to a set containing the empty set. Since the agent performs an incremental process, it is defined by a repeat-loop. The percept results Γ are sent to the queue Q. In order to take the observations Γ from the queue Q, the *MI_Agent* calls the *extractObservations* function.

In the following we assume that Γ represents an ordered set. $MAP(\Gamma, \mathcal{WR}, \mathcal{T})$ determines the most probable world of observations Γ with respect to a set of weighted rules $\mathcal{WR}$ and the Tbox $\mathcal{T}$. This function solves the maximum aposterior *MAP* inference problem defined in detail in [5]. It returns a vector W which consists of zeros and ones assigned to indicate whether the ground atoms of the considered world are true (positive) and false (negative), respectively. The function $\mathit{select}(W, \Gamma)$ then selects the positive assertions in the input Abox Γ using the bit vector W. The selected positive assertions are the assertions which require explanations. The *select* operation returns as output an Abox Γ' which has the following characteristic: $\Gamma' \subseteq \Gamma$. The determination of the most probable world by the *MAP* function and the selection of the positive assertions is also carried out on $\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0$.

In the next step, a set of forward chaining rules $\mathcal{FR}$ is applied to $\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0$. The generated assertions in this process are added to the to $\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0$. In the next step, only the consistent Aboxes are selected and the inconsistent Aboxes are removed. Afterwards, the *Interpret* function is called to determine the new agenda $\mathfrak{A}''$, the new interpretation Abox *NewI* and the Abox differences Δ_1 and Δ_2 for additions and omissions among *CurrentI* and *NewI*. Afterwards, the *CurrentI* is set to the *NewI* and the *MI_Agent* function communicates the Abox differences Δ_1 and Δ_2 to the *Partners*. The *manage_agenda* function is also called. This function is explained in [5]. The termination condition of the *MI_Agent* function is that the *Die*() function is true. Note that the *MI_Agent* waits in the function call $\mathit{extractObservations}(Q)$ if $Q = \emptyset$.

The $\mathit{manage\_agenda}(\mathfrak{A})$ function is called in the *MI_Agent* function to improve its performance. The agent can eliminate, shrink, or combine Aboxes.

After presenting the above algorithms, the mentioned unanswered questions can be discussed. A reason for performing the interpretation process and explaining the fiat assertions is that the probability of $P(\mathcal{A}, \mathcal{A}', \mathcal{R}, \mathcal{WR}, \mathcal{T})$ will increase through the interpretation process. In other words, by explaining the observations the agent's belief to the percepts will increase. This shows a new perspective for performing the interpretation process. The answer to the question whether there is any measure for stopping the interpretation process, is indeed positive. This is expressed by $\mid p_i - p_{i-1} \mid < \frac{\epsilon}{i}$ which is the termination condition Ξ of the algorithm. The reason for selecting $\frac{\epsilon}{i}$ and not ϵ as the upper limit for the termination condition is to terminate the oscillation behaviour of the results. In other words, the precision interval is tightened step by step during the interpretation process. In Section 3, we discuss an example for interpreting a single video shot.

3 Preference-Based Video Shot Interpretation

One of the main innovations introduced in the previous section, namely the introduction of a probabilistic preference measure to control the space of possible interpretations, is demonstrated here using examples from an environmental domain.

We have to mention that this example is not constructed to show the possible branchings through the interpretation process. The purpose of this example is to show how the probabilities of the most probable world of observations $P(\mathcal{A}_0, \mathcal{A}, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ behave during the interpretation process.

At the beginning of this example, the **signature** of the knowledge base is presented. The set of all concept names **CN** is divided into two disjoint sets **Events** and **PhysicalThings** such that

CN = **Events**$\cup$**PhysicalThings** where these two sets are defined as follows:

Events $= \{CarEntry, EnvConference, EnvProt, HealthProt\}$

PhysicalThings $= \{Car, DoorSlam, Building, Environment, Agency\}$

EnvConference, *EnvProt* and *HealthProt* denote respectively environmental conference, environmental protection and health protection. The set of role names **RN** is defined as follows:

$$\mathbf{RN} = \{Causes, OccursAt, HasAgency, HasTopic, HasSubject, HasObject, HasEffect, HasSubEvent, HasLocation\}$$

The set of individual names **IN** is defined as follows:

$$\mathbf{IN} = \{C_1, DS_1, ES_1, Ind_{42}, Ind_{43}, Ind_{44}, Ind_{45}, Ind_{46}, Ind_{47}, Ind_{48}\}$$

In the following, the set of the forward chaining rules $\mathcal{FR}$ is defined:

$$\begin{aligned}\mathcal{FR} = \{&\forall x \;\; CarEntry(x) \rightarrow \exists y \;\; Building(y), OccursAt(x, y),\\ &\forall x \;\; EnvConference(x) \rightarrow \exists y \;\; Environment(y), HasTopic(x, y),\\ &\forall x \;\; EnvProt(x) \rightarrow \exists y \;\; Agency(y), HasAgency(x, y)\}\end{aligned}$$

Similarly, the set of backward chaining rules $\mathcal{BR}$ is given as follows:

$$\begin{aligned}\mathcal{BR} = \{&Causes(x, y) \leftarrow CarEntry(z), HasObject(z, x), HasEffect(z, y), Car(x), DoorSlam(y),\\ &OccursAt(x, y) \leftarrow EnvConference(z), HasSubEvent(z, x), HasLocation(z, y), CarEntry(x), Building(y),\\ &HasTopic(x, y) \leftarrow EnvProt(z), HasSubEvent(z, x), HasObject(z, y), EnvConference(x), Environment(y),\\ &HasAgency(x, y) \leftarrow HealthProt(z), HasObject(z, x), HasSubject(z, y), EnvProt(x), Agency(y)\}\end{aligned}$$

In the following, a set of weighted rules $\mathcal{WR}$ is defined where the weight of each rule is 5:

$$\begin{aligned}\mathcal{WR} = \{&5 \; \forall x, y, z \;\; CarEntry(z) \wedge HasObject(z, x) \wedge HasEffect(z, y) \rightarrow Car(x) \wedge DoorSlam(y) \wedge Causes(x, y),\\ &5 \; \forall x, y, z \;\; EnvConference(z) \wedge HasSubEvent(z, x) \wedge HasLocation(z, y) \rightarrow\\ &\qquad CarEntry(x) \wedge Building(y) \wedge OccursAt(x, y),\\ &5 \; \forall x, y, z \;\; EnvProt(z) \wedge HasSubEvent(z, x) \wedge HasObject(z, y) \rightarrow\end{aligned}$$

$$EnvConference(x) \wedge Environment(y) \wedge HasTopic(x, y),$$
$$5\ \forall x, y, z\ HealthProt(z) \wedge HasObject(z, x) \wedge HasSubject(z, y) \rightarrow$$
$$EnvProt(x) \wedge Agency(y) \wedge HasAgency(x, y)\}$$

The selected value for ϵ in this example is 0.05. In the following, Δ_1 and Δ_2 denote respectively the set of assertions hypothesized by a forward chaining rule and the set of assertions generated by a backward chaining rule at each interpretation level. Let us assume that the media interpretation agent receives the following weighted Abox $\mathcal{A}$:

$$\mathcal{A} = \{1.3\ Car(C_1), 1.2\ DoorSlam(DS_1), -0.3\ EngineSound(ES_1), Causes(C_1, DS_1)\}$$

The first applied operation to $\mathcal{A}$ is the MAP function which returns the bit vector $W = \langle 1, 1, 0, 1\rangle$. By applying the *select* function to W and the input Abox $\mathcal{A}$, the assertions from the input Abox $\mathcal{A}$ are selected that correspond to the positive events in W. Additionally, the assigned weights to the positive assertions are also taken from the input Abox $\mathcal{A}$. In the following, Abox A_0 is depicted which contains the positive assertions:

$$\mathcal{A}_0 = \{1.3\ Car(C_1), 1.2\ DoorSlam(DS_1), Causes(C_1, DS_1)\}$$

At this step, $p_0 = P(\mathcal{A}_0, \mathcal{A}_0, \mathcal{R}, \mathcal{WR}, \mathcal{T}) = 0.755$. Since no appropriate forward chaining rule from $\mathcal{FR}$ is applicable to Abox $\mathcal{A}_0$, $\Delta_1 = \emptyset$ and as a result $\mathcal{A}_0 := \mathcal{A}_0 \cup \emptyset$. The next step is the execution of the *backward_chain* function where the next backward chaining rule from $\mathcal{BR}$ can be applied to Abox $\mathcal{A}_0$:

$$Causes(x, y) \leftarrow CarEntry(z), HasObject(z, x), HasEffect(z, y), Car(x), DoorSlam(y)$$

Consequently, by applying the above rule the next set of assertions is hypothesized:

$$\Delta_2 = \{CarEntry(Ind_{42}), HasObject(Ind_{42}, C_1), HasEffect(Ind_{42}, DS_1)\}$$

which are considered as strict assertions. Consequently, $\mathcal{A}_1$ is defined as follows: $\mathcal{A}_1 := \mathcal{A}_0 \cup \Delta_2$.
In the above Abox, $p_1 = P(\mathcal{A}_0, \mathcal{A}_1, \mathcal{R}, \mathcal{WR}, \mathcal{T}) = 0.993$. As it can be seen, $p_1 > p_0$ i.e. $P(\mathcal{A}_0, \mathcal{A}_i, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ used in Ξ increases by adding the new hypothesized assertions. This shows that the new assertions are considered as additional support. The termination condition of the algorithm is not fulfilled therefore the algorithm continues processing. At this level, it is still not known whether Abox $\mathcal{A}_1$ can be considered as the final interpretation Abox. Thus, this process is continued with another level. Consider the next forward chaining rule:

$$\forall x\ \ CarEntry(x) \rightarrow \exists y\ \ Building(y), OccursAt(x, y)$$

By applying the above rule, the next set of assertions is generated, namely:

$$\Delta_1 = \{Building(Ind_{43}), OccursAt(Ind_{42}, Ind_{43})\}$$

The generated assertions are also considered as strict assertions. In the following, the expanded Abox $\mathcal{A}_1$ is defined as follows: $\mathcal{A}_1 := \mathcal{A}_1 \cup \Delta_1$.

Let us assume the next backward chaining rule from $\mathcal{BR}$:

$$OccursAt(x,y) \leftarrow EnvConference(z), HasSubEvent(z,x), HasLocation(z,y), CarEntry(x), Building(y)$$

Consequently, by applying the above abduction rule the next set of assertions is hypothesized:

$\Delta_2 = \{EnvConference(Ind_{44}), HasSubEvent(Ind_{44}, Ind_{42}), HasLocation(Ind_{44}, Ind_{43})\}$

which are considered as strict assertions. Consequently, $\mathcal{A}_2 := \mathcal{A}_1 \cup \Delta_2$.

In the above Abox, $p_2 = P(\mathcal{A}_0, \mathcal{A}_2, \mathcal{R}, \mathcal{WR}, \mathcal{T}) = 0.988$. As it can be seen, $p_2 < p_1$ i.e.
$P(\mathcal{A}_0, \mathcal{A}_i, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ decreases slightly by adding the new hypothesized assertions. Since the termination condition of the algorithm is fulfilled, Abox $\mathcal{A}_1$ can be considered as the final interpretation Abox. To realize how the further behaviour of the probabilities is, this process is continued for the sake of illustration. Consider the next forward chaining rule from $\mathcal{FR}$:

$$\forall x \;\; EnvConference(x) \rightarrow \exists y \;\; Environment(y), HasTopic(x,y)$$

By applying the above rule, new assertions are generated.

$$\Delta_1 = \{Environment(Ind_{45}), HasTopic(Ind_{44}, Ind_{45})\}$$

In the following, the expanded Abox $\mathcal{A}_2$ is defined: $\mathcal{A}_2 := \mathcal{A}_2 \cup \Delta_1$.

Consider the next backward chaining rule from $\mathcal{BR}$:

$$HasTopic(x,y) \leftarrow EnvProt(z), HasSubEvent(z,x), HasObject(z,y), EnvConference(x), Environment(y)$$

By applying the above abduction rule, the following set of assertions is hypothesized:

$$\Delta_2 = \{EnvProt(Ind_{46}), HasSubEvent(Ind_{46}, Ind_{44}), HasObject(Ind_{46}, Ind_{45})\}$$

which are considered as strict assertions. In the following, $\mathcal{A}_3$ is defined as follows $\mathcal{A}_3 := \mathcal{A}_2 \cup \Delta_2$.
In the above Abox $\mathcal{A}_3$, $p_3 = P(\mathcal{A}_0, \mathcal{A}_3, \mathcal{R}, \mathcal{WR}, \mathcal{T}) = 0.99$. As it can be seen, $p_3 > p_2$, i.e.

$P(\mathcal{A}_0, \mathcal{A}_i, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ increases slightly by adding the new hypothesized assertions.

Consider the next forward chaining rule:

$$\forall x \;\; EnvProt(x) \rightarrow \exists y \;\; Agency(y), HasAgency(x,y)$$

By applying the above rule, the next assertions are generated:

$$\Delta_1 = \{Agency(Ind_{47}), HasAgency(Ind_{46}, Ind_{47})\}$$

As a result, the expanded Abox $\mathcal{A}_3$ is presented as follows: $\mathcal{A}_3 := \mathcal{A}_3 \cup \Delta_1$.

Let us consider the next backward chaining rule from $\mathcal{BR}$:

$$HasAgency(x,y) \leftarrow HealthProt(z), HasObject(z,x), HasSubject(z,y), EnvProt(x), Agency(y)$$

Consequently, new assertions are hypothesized by applying the above abduction rule, namely:

$$\Delta_2 = \{HealthProt(Ind_{48}), HasObject(Ind_{48}, Ind_{46}), HasSubject(Ind_{48}, Ind_{47})\}$$

which are considered as strict assertions. Consequently, $\mathcal{A}_4$ is defined as follows: $\mathcal{A}_4 := \mathcal{A}_3 \cup \Delta_2$.

In the above Abox, $p_4 = P(\mathcal{A}_0, \mathcal{A}_4, \mathcal{R}, \mathcal{WR}, \mathcal{T}) = 0.985$. As it can be seen, $p_4 < p_3$, i.e.

$P(\mathcal{A}_0, \mathcal{A}_i, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ decreases slightly by adding the new hypothesized assertions.

Discussion of the Results

The determined probability values $P(\mathcal{A}_0, \mathcal{A}_i, \mathcal{R}, \mathcal{WR}, \mathcal{T})$ of this example are summarized in the next table which shows clearly the behaviour of the probabilities stepwise after performing the interpretation process. In this table, variable i denotes the successive levels of the interpretation process. In this example, the interpretation process is consecutively performed four times. As it can be seen, through the first interpretation level the probability p_1 increases strongly in comparison to p_0. By performing the second, third and the forth interpretation levels, the probability values decrease slightly in comparison to p_1. This means no significant changes can be seen in the results. In other words, the determination of $\mathcal{A}_3$ and $\mathcal{A}_4$ were not required at all. But the determination of $\mathcal{A}_2$ was required to realize the slight difference $|p_2 - p_1| < \frac{\epsilon}{2}$. Consequently, Abox $\mathcal{A}_1$ is considered as the final interpretation Abox.

i	*Abox* $\mathcal{A}_i$	$p_i = P(\mathcal{A}_0, \mathcal{A}_i, \mathcal{R}, \mathcal{WR}, \mathcal{T})$
0	$\mathcal{A}_0$	$p_0 = 0.755$
1	$\mathcal{A}_1$	$p_1 = 0.993$
2	$\mathcal{A}_2$	$p_2 = 0.988$
3	$\mathcal{A}_3$	$p_3 = 0.99$
4	$\mathcal{A}_4$	$p_4 = 0.985$

4 Summary

For multimedia interpretation, a semantically well-founded formalization is required. In accordance with previous work, a well-founded abduction-based approach is pursued. Extending previous work, abduction is controlled by probabilistic knowledge, and it is done in terms of first-order logic. Rather than merely using abduction for computing explanations with which observations are entailed, the approach presented in this paper, uses a probabilistic logic to motivate the explanation endeavor by increasing the belief in the observations. Hence, there exists a certain utility for an agent for the computational resources it spends for generating explanations. Thus, we have presented a first attempt to more appropriately model a media interpretation agent and evaluated it using a detailed example.

References

1. Castano, S., Espinosa, S., Ferrara, A., Karkaletsis, V., Kaya, A., Möller, R., Montanelli, S., Petasis, G., Wessel, M.: Multimedia interpretation for dynamic ontology evolution. Journal of Logic and Computation (2008)
2. Domingos, P., Richardson, M.: Markov logic: A unifying framework for statistical relational learning. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning, pp. 339–371. MIT Press, Cambridge (2007)
3. Espinosa-Peraldi, S., Kaya, A., Möller, R.: Formalizing multimedia interpretation based on abduction over description logic aboxes. In: Cuenca-Grau, B., Horrocks, I., Motik, B. (eds.) Proc. of the 22nd International Workshop on Description Logics, DL 2009 (2010)
4. Espinosa-Peraldi, S., Kaya, A., Möller, R.: Logical formalization of multimedia interpretation. In: Tsatsaronis, G., Paliouras, G., Spyropoulos, C.D. (eds.) Knowledge-Driven Multimedia Information Extraction and Ontology Evolution. LNCS, vol. 6050. Springer, Heidelberg (2010)
5. Gries, O., Möller, R., Nafissi, A., Rosenfeld, M., Sokolski, K., Wessel, M.: A probabilistic abduction engine for media interpretation. Technical report, Hamburg University of Technology (2010), `http://www.sts.tu-harburg.de/tech-reports/2010/GMNR10.pdf`
6. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo (1988)
7. Poole, D.: Probabilistic horn abduction and bayesian networks. Artificial Intelligence 64(1), 81–129 (1993)

Secrecy-Preserving Query Answering for Instance Checking in $\mathcal{EL}$

Jia Tao, Giora Slutzki, and Vasant Honavar

Iowa State University, Ames, IA, USA

Abstract. We consider the problem of answering queries against an $\mathcal{EL}$ knowledge base (KB) using secrets, whenever it is possible to do so without compromising secrets. We provide a polynomial time algorithm that, given an $\mathcal{EL}$ KB Σ, a set $\mathbb{S}$ of secrets to be protected and a query q, outputs "Yes" whenever $\Sigma \models q$ and the answer to q, together with the answers to any previous queries answered by the KB, does not allow the querying agent to deduce any of the secrets in $\mathbb{S}$. This approach allows more flexible information sharing than is possible with traditional access control mechanisms.

1 Introduction

The rapid expansion of the World Wide Web and the widespread use of distributed databases and networked information systems offer unprecedented opportunities for productive interaction and collaboration among individuals and organizations in virtually every area of human endeavor. However, the need to share information has to be balanced against the need to protect secrets. Such scenarios call for algorithms that can, given a knowledge base Σ and a set $\mathbb{S}$ of secrets (perhaps specified using some secrecy policy), answer queries against Σ, using secrets if necessary, whenever it is possible to do so without compromising secrets (See Example 1 in Sect. 2). Most existing approaches to information protection simply *forbid* the use of secret information in answering queries (See Sect. 4). The *privacy-preserving reasoning* framework introduced in [1] was motivated by the need to alleviate, at least in part, this limitation in the simple setting of *hierarchical* knowledge bases (KBs) under the *open world assumption* (OWA)[1]. Such KBs may contain scientific, medical, economic information, or military intelligence, etc. Our secrecy-preserving reasoning framework builds on, and substantially extends, the framework introduced by Bao et al. [1].

In general, the answer to a query q of the form $C(a)$ or $r(a, b)$ against a KB Σ can be "Yes" (i.e., q can be inferred from Σ), "No" ($\neg q$ can be inferred from Σ) or "Unknown" (e.g., because of the incompleteness of Σ). We assume cooperative as opposed to adversarial scenarios where the KB does not *lie*. However,

[1] Under the closed world assumption a statement that cannot be inferred from the KB to be true, is presumed to be false. Under the OWA, the truth of such a statement is presumed to be unknown, and *not necessarily* false.

whenever truthfully answering a query risks compromising secrets in $\mathbb{S}$, the reasoner associated with the KB is allowed to feign ignorance, i.e., answer the query as "Unknown". Given a set of secrets $\mathbb{S}$ (which need not be a subset of Σ), it is clear that, to protect $\mathbb{S}$, answers to queries in $\mathbb{S}$ will be "Unknown". However, in general, it is not sufficient to protect only $\mathbb{S}$ since truthful answers to certain queries (not in $\mathbb{S}$) may reveal information in $\mathbb{S}$. Therefore, we must protect a superset of $\mathbb{S}$, which we call an *envelope* of $\mathbb{S}$, such that the querying agent who has no access to the envelope will not be able to deduce any information in $\mathbb{S}$.

In this paper, we investigate secrecy-preserving query answering with $\mathcal{EL}$ [2], which is one of the simplest DLs that is both computationally tractable [3,4] and practically useful [2]. For example, the medical ontology SNOMED CT [5] and large parts of the medical ontology GALEN [6] can be expressed in $\mathcal{EL}$. We provide algorithms to answer queries against an $\mathcal{EL}$ KB that use, but not reveal, the information that is designated as secret. Because of the open world assumption and the fact that the language of $\mathcal{EL}$ does not include negation, the answer to a query can only be "Yes" or "Unknown".

To answer queries posed to the KB, we construct a *secrecy maintenance system* that consists of: a finite set of consequences of the KB Σ, denoted by $\mathcal{A}^*$, and a secrecy envelope $\mathbb{S} \subseteq \mathbb{E}_{\mathbb{S}} \subseteq \mathcal{A}^*$. The answer to a query q is censored by the reasoner if $q \in \mathbb{E}_{\mathbb{S}}$. It is easy to see that a secrecy envelope always exists. For instance, $\mathcal{A}^*$ constitutes an envelope for any secrecy set $\mathbb{S} \subseteq \mathcal{A}^*$. A key challenge is to *develop strategies that can be used by the KB to respond to queries as informatively as possible (i.e., using an envelope that is as small as possible) without compromising secrets that the KB is obliged to protect.* Unfortunately, computing a minimum envelope is NP-hard [7]. We compute $\mathcal{A}^*$ using the (usual) tableau expansion rules. To compute $\mathbb{E}_{\mathbb{S}}$, we introduce the following idea. From each original expansion rule, we construct a corresponding *inverse expansion rule*. We show that the inverted system of expansion rules generates an envelope of $\mathbb{S}$. To the best of our knowledge, the idea of constructing a secrecy envelope by inverting the tableau expansion rules is novel. Furthermore, we introduce a couple of useful optimizations that help reduce the size of an envelope.

2 Preliminaries

The non-logical signature of the $\mathcal{EL}$ description language includes three mutually disjoint sets: *concept names* $N_{\mathcal{C}}$, *role names* $N_{\mathcal{R}}$ and *individual names* $N_{\mathcal{O}}$. The syntax of $\mathcal{EL}$ is defined by specifying *expressions* and *formulae.* $\mathcal{EL}$ expressions consist of $N_{\mathcal{R}}$ and the set of *concepts* $\mathcal{C}$ recursively defined as follows:

$$C, D \longrightarrow A \mid \top \mid C \sqcap D \mid \exists r.C$$

where $A \in N_{\mathcal{C}}$, $\top$ is the *top symbol*, $C, D \in \mathcal{C}$ and $r \in N_{\mathcal{R}}$. In this paper we consider three kinds of $\mathcal{EL}$ formulae: *assertions* of the form $C(a)$ or $r(a, b)$, *definitions* of the form $A \doteq D$ and *general concept inclusions (GCI)* of the form $C \sqsubseteq D$ where $a, b \in N_{\mathcal{O}}$, $C, D \in \mathcal{C}$, $r \in N_{\mathcal{R}}$ and $A \in N_{\mathcal{C}}$.

The semantics of $\mathcal{EL}$ is specified by means of an *interpretation* $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ where Δ is a non-empty domain and $\cdot^{\mathcal{I}}$ is a function that maps each individual name to an element in Δ, each concept name to a subset of Δ and each role name to a subset of $\Delta \times \Delta$. The interpretation of concept expressions is extended recursively: for $r \in N_{\mathcal{R}}$ and $C, D \in \mathcal{C}$: $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ and $(\exists r.C)^{\mathcal{I}} = \{a \in \Delta \mid \exists b \in \Delta : (a, b) \in r^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$.

A finite non-empty set of assertions is called an *ABox.* A finite set of definitions and GCIs is called a *TBox.* An ABox $\mathcal{A}$ and a TBox $\mathcal{T}$ whose concepts and roles belong to the language $\mathcal{EL}$ form an $\mathcal{EL}$-knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$. A TBox $\mathcal{T}$ is *normalized* [3] if $\mathcal{T}$ contains only GCIs all of which are of one of the following forms: $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq B$, $A \sqsubseteq \exists r.B$ or $\exists r.A \sqsubseteq B$ where $A, A_1, A_2, B \in N_{\mathcal{C}} \cup \{\top\}$. It was shown that transforming a TBox into such a normal form can be accomplished in polynomial time [3]. From now on, we will assume that all the TBoxes are in normal form. By N_{Σ} (resp. $\mathcal{O}_{\Sigma}$) we denote the set of all symbols (resp. individual names) occurring in Σ. Note that $\mathcal{O}_{\Sigma} \subset N_{\mathcal{O}} \cap N_{\Sigma}$ and $N_{\Sigma} \setminus \mathcal{O}_{\Sigma} \subset N_{\mathcal{C}} \cup N_{\mathcal{R}}$.

Definition 1. *Let $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ be a knowledge base, $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ an interpretation, $C, D \in \mathcal{C}$, $r \in N_{\mathcal{R}}$ and $a, b \in N_{\mathcal{O}}$. $\mathcal{I}$* satisfies *$C(a)$, $r(a, b)$, or $C \sqsubseteq D$ if, respectively, $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, or $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ is a* model *of Σ if it satisfies all the assertions in $\mathcal{A}$ and all the GCIs in $\mathcal{T}$. Let α be an assertion or a GCI. We say that Σ* entails *α, written as $\Sigma \models \alpha$, if all models of Σ satisfy α.*

Example 1. (a simplified version adapted from [8]) Given a KB $\Sigma_1 = \langle \mathcal{A}_1, \mathcal{T}_1 \rangle$ that contains information on the patients, their health history, the prescriptions that they get from the physicians and their insurance information. Suppose that Jane*'s* mother Jill had breast cancer and that Jane tests positive for BRCA1 mutation which is linked to an increased risk of breast cancer. To reduce the breast cancer risk, Jane was prescribed a certain drug. Jane purchases the medications from her pharmacy and wants to get reimbursed for the cost of her prescription by her insurance company. If her insurance company finds out that she has tested positive for BRCA1 mutation or that she has been prescribed certain drug(s) for breast cancer, Jane risks losing her health insurance. The scenario can be formally specified in the DL $\mathcal{EL}$ as follows:

1. $\exists$is_child.A $\sqsubseteq$ CancerRisk	7. A $\sqsubseteq$ HasCancer
2. HasMutBRCA1 $\sqsubseteq$ $\exists$has_pres.CancerDrug	8. Woman $\sqcap$ HasCancer $\sqsubseteq$ A
3. $\exists$has_pres.CancerDrug $\sqsubseteq$ CancerRisk	9. Woman(Jill)
4. $\exists$has_pres.CoveredDrug $\sqsubseteq$ Reimburse	10. HasCancer(Jill)
5. CancerDrug $\sqsubseteq$ CoveredDrug	11. is_child(Jane, Jill)
6. A $\sqsubseteq$ Woman	12. HasMutBRCA1(Jane)

The GCIs 1-8 form a subset of $\mathcal{T}_1$ (in normal form) and the assertions 9-12 form a subset of $\mathcal{A}_1$. In order for Jane to get reimbursed, when the query Reimburse(Jane) is posed to the KB, the answer should be "Yes". However, in order to protect Jane's privacy, the query CancerRisk(Jane) should be answered "Unknown". ■

3 The Secrecy-Preserving Query Answering

Problem Statement: Given a knowledge base Σ and a finite secrecy set $\mathbb{S}$, the basic goal is to answer queries while preserving secrecy. As shown in Example 1, to protect Jane's privacy, the query CancerRisk(Jane) should be answered "Unknown". However, by only keeping CancerRisk(Jane) secret, the fact that Jane has cancer risk can still be inferred by statements 12, 2 and 3. Therefore, the secrecy-preserving query answering problem is to find a superset of $\mathbb{S}$, which we call the *secrecy envelope* of $\mathbb{S}$, denoted by $\mathbb{E}_{\mathbb{S}}$, so that by protecting $\mathbb{E}_{\mathbb{S}}$, the querying agent cannot conclude anything in $\mathbb{S}$. Because of the OWA, when the answer to a query is "Unknown", the querying agent is not able to distinguish between (a) the answer to the query is truly unknown, or (b) the answer is being protected for reasons of secrecy.

The framework contains following components. We assume a KB $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$, a reasoner $\mathfrak{R}$ that is complete, and a secrecy set $\mathbb{S}$ consisting of a finite set of assertions that contain only symbols from N_{Σ}. $\mathfrak{R}$ is used to answer queries by checking whether the query can be inferred from Σ and if it can, whether answering "Yes" will reveal secrets from $\mathbb{S}$. The specific tasks are:

- To compute the set SubC of sub-expressions of all concepts and roles appearing in Σ or $\mathbb{S}$.
- To compute the set of all assertional consequences of Σ restricted to SubC. This set is called the *assertional closure of* Σ and it is denoted by $\mathcal{A}^*$. We assume that $\mathbb{S} \subseteq \mathcal{A}^*$.
- To compute the secrecy envelope $\mathbb{S} \subseteq \mathbb{E}_{\mathbb{S}} \subseteq \mathcal{A}^*$, a set of assertions which if truthfully answered, may reveal some secret(s) in $\mathbb{S}$.
- To answer queries. If a query cannot be inferred from Σ, the answer is simply "Unknown". If it can be inferred and it is not in $\mathbb{E}_{\mathbb{S}}$, the answer is "Yes"; otherwise, the answer is "Unknown".

We also assume that the querying agent (i) asks queries of the form $C(a)$ or $r(a, b)$; (ii) has computational access only to the signature of the knowledge base, i.e., its queries are over N_{Σ}; and (iii) has the same reasoning capacity as $\mathfrak{R}$ (Since we assume that $\mathfrak{R}$ is complete, this is not a restriction.); (iv) may log the history of all the answers to its queries and draw conclusions from it; and (v) has access to the TBox $\mathcal{T}$.

$\mathcal{A}^*$ and $\mathbb{E}_{\mathbb{S}}$ form a *secrecy maintenance system*. Note that the both are restricted to SubC. Once $\mathcal{A}^*$ and $\mathbb{E}_{\mathbb{S}}$ have been computed, if $C \in \mathit{SubC}$, $\mathfrak{R}$ can answer the query $C(a)$ in linear time depending on its membership of $\mathcal{A}^*$ and $\mathbb{E}_{\mathbb{S}}$. Otherwise, we need to expand SubC by adding sub-expressions of C that are not in SubC and update the consequences $\mathcal{A}^*$ as well as $\mathbb{E}_{\mathbb{S}}$ accordingly.

3.1 Initializing Secrecy Maintenance System

Computing SubC**:** The set of certain sub-expressions of all the concepts and roles appearing in Σ or $\mathbb{S}$, is defined as follows:

if $C(a) \in \mathcal{A} \cup \mathbb{S}$, then $C \in Sub\mathcal{C}$; if $C \sqsubseteq D \in \mathcal{T}$, then $\{C, D\} \subseteq Sub\mathcal{C}$;
if $r(a, b) \in \mathcal{A} \cup \mathbb{S}$, then $r \in Sub\mathcal{C}$; if $\exists r.C \in Sub\mathcal{C}$, then $\{r, C\} \subseteq Sub\mathcal{C}$;
if $C_1 \sqcap \cdots \sqcap C_k \in Sub\mathcal{C}(C_i \in N_\mathcal{C}$ or $C_i = \exists r.C)$, then $C_i \in Sub\mathcal{C}(1 \leq i \leq k)$;
if $\exists r.C \in Sub\mathcal{C}$ and $C \sqsubseteq D \in \mathcal{T}$ or $D \sqsubseteq C \in \mathcal{T}$, then $\exists r.D \in Sub\mathcal{C}$.

Note that $Sub\mathcal{C}$ does not contain all the sub-expressions of concepts appearing in Σ or $\mathbb{S}$. If a query $C(a)$ comes along where $C \notin Sub\mathcal{C}$, it will be added into $Sub\mathcal{C}$. As such, the secrecy maintenance system is built up gradually depending on the history of queries. Also note that the initial size of $Sub\mathcal{C}$ is linear in the size of the knowledge base Σ plus the size of the secrecy set $\mathbb{S}$.

Computing $\mathcal{A}^*$: The ABox $\mathcal{A}^*$ is initialized as $\mathcal{A}$ and expanded by recursively applying assertion expansion rules listed in Fig. 1. We say that $\mathcal{A}^*$ is *assertionally closed* or that it is an *assertional closure of* Σ if no assertion expansion rule is applicable. The set of all the individual names appearing in $\mathcal{A}^*$ is denoted by $\mathcal{O}^*$. It is initialized as $\mathcal{O}_\Sigma$ and is expanded with applications of the $\exists_2^\mathcal{A}$-rule. An individual a is said to be *fresh* (at a particular time during the expansion process) if $a \in N_\mathcal{O} \setminus \mathcal{O}^*$ (at that time). An individual $a \in \mathcal{O}^*$ is *blocked* by an individual $b \in \mathcal{O}^*$ if $a \in \mathcal{O}^* \setminus \mathcal{O}_\Sigma$, b is either in $\mathcal{O}_\Sigma$ or b was picked earlier than a (during the expansion process), and $\{C \mid C(a) \in \mathcal{A}^*\} \subseteq \{C' \mid C'(b) \in \mathcal{A}^*\}$. Recall that we have assumed that the querying agent has computational access only to the signature of the knowledge base. In particular, the querying agent cannot ask any queries that involve individual names in $\mathcal{O}^* \setminus \mathcal{O}_\Sigma$. This is referred to as *Hidden Names Assumption* (HNA).

$\sqcap_1^\mathcal{A}$ -rule:	if $C_1 \sqcap \cdots \sqcap C_k(a) \in \mathcal{A}^*$ and $C_i(a) \notin \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{C_i(a)\}$ where $1 \leq i \leq k$;
$\sqcap_2^\mathcal{A}$ -rule:	if $\{C_1(a), ..., C_k(a)\} \subseteq \mathcal{A}^*, C_1 \sqcap \cdots \sqcap C_k \in Sub\mathcal{C}$ and $C_1 \sqcap \cdots \sqcap C_k(a) \notin \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{C_1 \sqcap \cdots \sqcap C_k(a)\}$;
$\exists_1^\mathcal{A}$ -rule:	if $\{r(a, b), C(b)\} \subseteq \mathcal{A}^*, \exists r.C \in Sub\mathcal{C}$ and $\exists r.C(a) \notin \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{\exists r.C(a)\}$;
$\exists_2^\mathcal{A}$ -rule:	if $\exists r.C(a) \in \mathcal{A}^*, a$ is not blocked and $\forall b \in \mathcal{O}^*, \{r(a, b), C(b)\} \nsubseteq \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{r(a, c), C(c)\}$ where c is fresh, and $\mathcal{O}^* := \mathcal{O}^* \cup \{c\}$;
$\sqsubseteq^\mathcal{T}$ -rule:	if $C(a) \in \mathcal{A}^*, C \sqsubseteq D \in \mathcal{T}$ and $D(a) \notin \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{D(a)\}$;

Fig. 1. Assertion Expansion Rules

We denote by Λ the tableau algorithm which nondeterministically applies assertion expansion rules until no further applications are possible. Since each expansion rule can be applied polynomially many times (in the size of $Sub\mathcal{C}$), the computation of $\mathcal{A}^*$ can be done in polynomial time. When an execution of Λ terminates, we have an assertionally closed ABox $\mathcal{A}^*$. The soundness and completeness of the Λ-*tableau algorithm* are proved in [7].

Ignoring the issue of secrecy, we point out a difference between the reasoning of the KB reasoner $\mathfrak{R}$ and that of the querying agent. Consider the assertion $\exists r.C(a) \in \mathcal{A}^*$ when a is not blocked and there does not exist $b \in \mathcal{O}_\Sigma$ for which

$\{r(a,b), C(b)\} \subseteq \mathcal{A}^*$. In this case $\mathfrak{R}$ picks a fresh individual name $c \notin \mathcal{O}_\Sigma$ as a witness for the inclusion $\exists r.C(a) \in \mathcal{A}^*$. The querying agent only knows the existence of the witness individual and not the individual name itself. Of course, for its own reasoning process, the querying agent may pick any individual name in $N_\mathcal{O} \setminus \mathcal{O}_\Sigma$, say d, and then force $r(a,d)$ and $C(d)$ to be consequences of Σ. Clearly, the reasoner $\mathfrak{R}$ and the querying agent are not aware of each other's "fresh" individual names. To differentiate the assertional closure of the KB reasoner $\mathfrak{R}$ from the reasoning of the querying agent, we will use $\cdot^+$ to denote the latter.

Computing the Secrecy Envelopes: We define the secrecy envelope $\mathbb{E}_\mathbb{S}$ such that if the reasoner $\mathfrak{R}$ answers every query in $\mathbb{E}_\mathbb{S}$ with "Unknown" and every query in $\mathcal{A}^* \setminus \mathbb{E}_\mathbb{S}$ with "Yes", the querying agent will not be able to deduce any assertions in $\mathbb{S}$.

Definition 2. *Given a knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ and a finite secrecy set $\mathbb{S} \subseteq \mathcal{A}^*$, a* secrecy envelope *of $\mathbb{S}$, denoted by $\mathbb{E}_\mathbb{S}$, is a superset $\mathbb{S} \subseteq \mathbb{E}_\mathbb{S} \subseteq \mathcal{A}^*$ such that $(\mathcal{A}^* \setminus \mathbb{E}_\mathbb{S})^+ \cap \mathbb{S} = \emptyset$ where $(\mathcal{A}^* \setminus \mathbb{E}_\mathbb{S})^+$ is the assertional closure of the knowledge base $\langle \mathcal{A}^* \setminus \mathbb{E}_\mathbb{S}, \mathcal{T} \rangle$ for the querying agent.*

To answer queries as informatively as possible, we aim to make $\mathbb{E}_\mathbb{S}$ as small as possible. Unfortunately, to compute a minimum envelope is hard. Specifically, the decision version of the problem of computing minimum envelopes is NP-complete [7]. In what follows, we provide an algorithm that computes envelopes. Utilizing the HNA, we further optimize the algorithm to result a smaller envelope. To compute an envelope, we introduce the novel idea of *inverting assertion expansion rules*. For $\mathcal{EL}$ with TBox, we have five assertion expansion rules as listed in Fig. 1. For each assertion expansion rule, the resulting inverse rule is named by changing the superscript in the name of the original rule to $\mathbb{S}$. These inversion rules are called $\mathfrak{R}$-*secrecy closure rules* and are listed in Fig. 2. In Fig. 2, $\mathcal{A}^*$ is assumed to have been computed previously; $\mathbb{E}$ is initialized to $\mathbb{S}$, and expanded by using $\mathfrak{R}$-secrecy closure rules.

$\sqcap_1^\mathbb{S}$ -rule:	if $C_1 \sqcap \cdots \sqcap C_k(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $\{C_1(a), ..., C_k(a)\} \cap \mathbb{E} \neq \emptyset$, then $\mathbb{E} := \mathbb{E} \cup \{C_1 \sqcap \cdots \sqcap C_k(a)\}$;
$\sqcap_2^\mathbb{S}$ -rule:	if $C_1 \sqcap \cdots \sqcap C_k(a) \in \mathbb{E}$ and $\{C_1(a), ..., C_k(a)\} \cap \mathbb{E} = \emptyset$, then $\mathbb{E} := \mathbb{E} \cup \{C_i(a)\}$ where $1 \leq i \leq k$;
$\exists_1^\mathbb{S}$ -rule:	if $\exists r.C(a) \in \mathbb{E}$ and $\{r(a,b), C(b)\} \subseteq \mathcal{A}^* \setminus \mathbb{E}$ with $b \in \mathcal{O}^*$, then $\mathbb{E} := \mathbb{E} \cup \{r(a,b)\}$ or $\mathbb{E} := \mathbb{E} \cup \{C(b)\}$;
$\exists_2^\mathbb{S}$ -rule:	if $\exists r.C(a) \in \mathcal{A}^* \setminus \mathbb{E}$, and for every $b \in \mathcal{O}^*$ with $\{r(a,b), C(b)\} \subseteq \mathcal{A}^*$, we have $\{r(a,b), C(b)\} \cap \mathbb{E} \neq \emptyset$, then $\mathbb{E} := \mathbb{E} \cup \{\exists r.C(a)\}$;
$\sqsubseteq^\mathbb{S}$ -rule:	if $D(a) \in \mathbb{E}, C \sqsubseteq D \in \mathcal{T}$ and $C(a) \in \mathcal{A}^* \setminus \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{C(a)\}$.

Fig. 2. $\mathfrak{R}$-secrecy closure rules obtained by inverting rules in Fig. 1

We denote by $\Lambda_\mathbb{S}^\mathfrak{R}$ the tableau algorithm which nondeterministically applies the $\mathfrak{R}$-secrecy closure rules until no further rules are applicable. When no $\mathfrak{R}$-secrecy closure rule is applicable, we say that $\mathbb{E}$ is $\mathfrak{R}$-*closed*. It is easy to see that

$\Lambda_{\mathbb{S}}^{\mathfrak{R}}$ terminates in polynomial time in the size of its input. The following lemma and corollary show that $\Lambda_{\mathbb{S}}^{\mathfrak{R}}$ results an envelope. The proofs are available in [7].

Lemma 1. *Let $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ be a KB, $\mathbb{S} \subseteq \mathbb{E} \subseteq \mathcal{A}^*$ where $\mathbb{S}$ is the secrecy set and $\mathbb{E}$ is $\mathfrak{R}$-closed. Then (a) $\mathcal{A}^* \setminus \mathbb{E}$ is assertionally closed w.r.t. assertion expansion rules listed in Fig. 1, (b) $\mathbb{E}$ is a secrecy envelope of $\mathbb{S}$.*

It turns out that the $\Lambda_{\mathbb{S}}^{\mathfrak{R}}$ algorithm, although certainly producing an envelope, may actually result an envelope that is unnecessarily large. Specifically, even if $\exists_2^{\mathcal{A}}$-rule is applicable to $(\mathcal{A}^* \setminus \mathbb{E}_{\mathbb{S}})^+$, due to OWA, the querying agent can only conclude that there exists an individual d that is the witness for $\exists r.C(a)$ and that $d \notin \mathcal{O}_\Sigma$. However, by HNA, the querying agent has no computational access to individual names in $\mathcal{O}^* \setminus \mathcal{O}_\Sigma$. This provides a cue that when computing a secrecy envelope, the $\exists_2^{\mathbb{S}}$-rule, which inverts the $\exists_2^{\mathcal{A}}$-rule, is dispensable. The new set of secrecy closure rules, called *$\mathcal{Q}$-Secrecy Closure Rules*, includes only the $\sqcap_1^{\mathbb{S}}$-rule, the $\sqcap_2^{\mathbb{S}}$-rule, the $\sqsubseteq^{\mathbb{S}}$-rule and the $\exists_1^{\mathbb{S}}$-rule is replaced by an "optimized" version the $\exists^{\mathbb{S}}$-rule.

$\exists^{\mathbb{S}}$ -rule: if $\exists r.C(a) \in \mathbb{E}$ and $\{r(a,b), C(b)\} \subseteq \mathcal{A}^* \setminus \mathbb{E}$ with $b \in \mathcal{O}_\Sigma$,
then $\mathbb{E} := \mathbb{E} \cup \{r(a,b)\}$ or $\mathbb{E} := \mathbb{E} \cup \{C(b)\}$

We denote by $\Lambda_{\mathbb{S}}^{\mathcal{Q}}$ the tableau algorithm which nondeterministically and exhaustively applies the $\mathcal{Q}$-secrecy closure rules. The resulting $\mathbb{E}$ is said to be *$\mathcal{Q}$-closed.* It is clear that all executions of $\Lambda_{\mathbb{S}}^{\mathcal{Q}}$ terminate in polynomial time. Theorem 1 shows that $\Lambda_{\mathbb{S}}^{\mathcal{Q}}$ also results an envelope. Proofs can be found in [7].

Theorem 1. *Let $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ be a KB, $\mathbb{S} \subseteq \mathbb{E} \subseteq \mathcal{A}^*$ where $\mathbb{S}$ is the secrecy set and $\mathbb{E}$ is $\mathcal{Q}$-closed. Then $\mathbb{E}$ is a secrecy envelope of $\mathbb{S}$.*

Note that the whole initialization of the secrecy maintenance system (including computation of $Sub\mathcal{C}$, $\mathcal{A}^*$ and $\mathbb{E}_{\mathbb{S}}$) is easily seen to be doable in polynomial time in the size of the KB Σ plus the size of the given secrecy set $\mathbb{S}$.

3.2 Query Answering

In this section we assume that the three sets $Sub\mathcal{C}$, $\mathcal{A}^*$ and $\mathbb{E}_{\mathbb{S}}$ (the latter two, restricted to $Sub\mathcal{C}$) have been precomputed in the pre-query stage as described in Sect. 3.1. The computation of the answer to a query of the form $C(a)$ is given in Fig. 3. The input of the secrecy-preserving query answering procedure SPQA contains the TBox $\mathcal{T}$ in normal form, precomputed assertional closure $\mathcal{A}^*$, the query $C(a)$ and the precomputed secrecy envelope $\mathbb{E}_{\mathbb{S}}$. Since sub-expressions of C, denoted by $sub(C)$, need not be in $Sub\mathcal{C}$, Line 2 in the SPQA procedure expands $Sub\mathcal{C}$ by adding expressions in $sub(C) \setminus Sub\mathcal{C}$. The expanded $Sub\mathcal{C}$ will be used to update $\mathcal{A}^*$ by applying assertion expansion rules (Fig. 1) until none of them is applicable, as indicated in Line 2. As a consequence, there may be applicable $\mathcal{Q}$-secrecy closure rules, implying that $\mathbb{E}_{\mathbb{S}}$ may no longer be a secrecy envelope for $\mathbb{S}$. Therefore, we apply necessary secrecy closure rules exhaustively

```
SPQA($\mathcal{T}, \mathcal{A}^*, C(a), \mathbb{E}_{\mathbb{S}}$):
1.  if ($C \notin Sub\mathcal{C}$) {
2.      compute $sub(C)$; $Sub\mathcal{C} = Sub\mathcal{C} \cup sub(C)$; expand $\mathcal{A}^*$ to $Sub\mathcal{C}$;
3.      expand the secrecy envelope $\mathbb{E}_{\mathbb{S}}$ to $Sub\mathcal{C}$; }
4.  if ($C(a) \in \mathcal{A}^*$ and $C(a) \notin \mathbb{E}_{\mathbb{S}}$) return "Yes";
5.  else return "Unknown";
```

Fig. 3. Secrecy-preserving Query-answering Procedure

(Line 3). Clearly, a single invocation of the procedure SPQA takes polynomial time (in the sum of the sizes of its arguments).

For queries of the form $r(a, b)$, the procedure is much simpler: if $r(a, b) \in \mathcal{A} \setminus \mathbb{E}_{\mathbb{S}}$, then the answer is "Yes"; otherwise, the answer is "Unknown". Here $\mathbb{E}_{\mathbb{S}}$ is the current secrecy envelope.

Example 2. (Example [1], continued) Recall that we have a KB $\Sigma_1 = \langle \mathcal{A}_1, \mathcal{T}_1 \rangle$ and the secrecy set $\mathbb{S}_1 = \{$CancerRisk(Jane)$\}$. The assertional closure of Σ_1, denoted by $\mathcal{A}_1^*$, and one possible envelope $\mathbb{E}_{\mathbb{S}1}$ are listed below:
$\mathcal{A}_1^* = \mathcal{A}_1 \cup \{$ A(Jill), ∃is_child.A(Jane), CancerRisk(Jane), has_pres(Jane, a), ∃has_pres.CancerDrug(Jane), CancerDrug(a), CoveredDrug(a), ∃has_pres.CoveredDrug(Jane), Reimburse(Jane)$\}$.
$\mathbb{E}_{\mathbb{S}1} = \{$CancerRisk(Jane), is_child(Jane, Jill), HasMutBRCA1(Jane), ∃is_child.A(Jane), ∃has_pres.CancerDrug(Jane)$\}$.

If the querying agent asks the query Reimburse(Jane), Reimburse(Jane)$\in \mathcal{A}_1^* \setminus \mathbb{E}_{\mathbb{S}1}$, the answer to the query is "Yes". If the querying agent asks the query CancerRisk(Jane), since CancerRisk(Jane)$\in \mathcal{A}_1^* \cap \mathbb{E}_{\mathbb{S}1}$, the answer to the query is "Unknown". ■

4 Summary and Discussion

Summary: In this paper, we have introduced a logic-based framework for secrecy preserving query answering in $\mathcal{EL}$ knowledge bases. We have provided a polynomial time algorithm that, given an $\mathcal{EL}$ KB Σ, a set $\mathbb{S}$ of secrets to be protected and a query q, truthfully answers the query whenever: (i) $\Sigma \vDash q$ and (ii) the answer to q, together with the answers to any previous queries answered by the KB does not allow the querying agent to deduce any of the secrets in $\mathbb{S}$. Our approach exploits the open world semantics under which it is impossible for the querying agent to distinguish between an answer "Unknown" resulting because of incomplete knowledge of the KB or because of selective censoring of answers by the KB. Our secrecy-preserving reasoning framework builds on, and substantially extends, the privacy-preserving reasoning framework introduced by Bao et al. [1] which considered protecting class-subclass relationships in hierarchical ontologies.

Related Work: Most of the work in this area falls into four broad categories of access control mechanisms, information confinement, preventing disclosure of information of specific individuals and controlled query evaluation. In contrast, our approach permits the use of secrets in answering queries for a given KB when it is possible to do so without compromising secrets under the OWA. A detailed comparison can be found in [7].

Future Work: Some natural directions for future work include: (i) design of an efficient algorithm for computing a "tight" envelope for $\mathcal{EL}$ KBs, i.e., an envelope from which no statement can be dropped without risking the possibility of secrets being compromised (such an algorithm is of interest in light of the fact that our current algorithm is not guaranteed to produce a tight envelope and the fact that computing the minimum envelope is NP-hard); (ii) exploration of secrecy-preserving query answering algorithms in the case of more expressive e.g., $\mathcal{ALC}$, DL-Lite, and RDF KBs; (iii) investigation of secrecy-preserving query answering in settings with multiple querying agents, under various restrictions on communication among agents.

References

1. Bao, J., Slutzki, G., Honavar, V.: Privacy-preserving reasoning on the semantic web. In: Web Intelligence, pp. 791–797. IEEE Computer Society Press, Los Alamitos (2007)
2. Baader, F.: Terminological cycles in a description logic with existential restrictions. In: IJCAI'03: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 325–330. Morgan Kaufmann Publishers Inc., San Francisco (2003)
3. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, gci axioms, and - what else? In: de Mántaras, R.L., Saitta, L. (eds.) ECAI, pp. 298–302. IOS Press, Amsterdam (2004)
4. Krisnadhi, A., Lutz, C.: Data complexity in the el family of dls. In: Calvanese, D., Franconi, E., Haarslev, V., Lembo, D., Motik, B., Turhan, A.-Y., Tessaris, S. (eds.) Description Logics, CEUR Workshop Proceedings, vol. 250. CEUR-WS.org (2007)
5. Spackman, K.: Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. J. of the Amer. Med. Informatics Assoc. (2000)
6. Rector, A., Horrocks, I.: 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, pp. 321–325 (1997)
7. Tao, J., Slutzki, G., Honavar, V.: Secrecy-preserving query answering in el. Technical Report TR10-03a, Iowa State University, Ames, IA (2010)
8. Farkas, C., Brodsky, A., Jajodia, S.: Unauthorized inferences in semistructured databases. Information Sciences 176, 3269–3299 (2006)

Embeddings of Simple Modular Extended RDF

Carlos Viegas Damásio[1], Anastasia Analyti[2], and Grigoris Antoniou[3]

[1] CENTRIA, Departamento de Informática da Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
cd@di.fct.unl.pt

[2] Institute of Computer Science, FORTH-ICS, Crete, Greece
analyti@ics.forth.gr

[3] Institute of Computer Science, FORTH-ICS, and Department of Computer Science, University of Crete, Crete, Greece
antoniou@ics.forth.gr

Abstract. The Extended Resource Description Framework has been proposed to equip RDF graphs with weak and strong negation, as well as derivation rules, increasing the expressiveness of ordinary RDF graphs. In parallel, the Modular Web framework enables collaborative and controlled reasoning in the Semantic Web. In this paper we exploit the use of the Modular Web framework to specify the modular semantics for Extended Resource Description Framework ontologies.

1 Introduction

The Extended Resource Description Framework [5] (ERDF) provides a model theoretical semantics for RDF graphs allowing negative triples, and ontologies defined by first-order rules including the two forms of negation, weak and strong. However, no means of combining different ontologies is specified. The necessity of mechanisms to encapsulate and organize knowledge in the Semantic Web is essential [13,6,8,10], and the ERDF framework has been extended to allow the specification of import and export declarations of classes and properties, resulting in the Modular ERDF framework [3]. The semantics of the modular ERDF framework has also been defined model theoretically, but it was lacking a declarative rule-based semantics for implementing the system.

In parallel, the Modular Web Framework (MWeb) is a proposal to address the issues of programming-in-the-wide faced by the new Semantic Web rule-engines [2,4]. MWeb defines general constructs to allow sharing of knowledge in the Semantic Web provided by logic based knowledge bases, including scoped open and closed world assumptions with contextualized and global interpretation of predicates. The MWeb framework is constructed, compatible and based on Rule Interchange Format (RIF) guidelines fostering immediate integration with RDF [12]. MWeb provides two semantics designated MWebWFS and MWebAS with a solid theory based on the two major semantics of extended logic programming, respectively, Well-Founded Semantics with Explicit Negation [1] and Answer Sets [9]. A compiler of MWeb into XSB Prolog is available[1] making use of the tabling features to guarantee termination of recursive rules with negation. It provides separate interface and implementation of rulebases with modular and independent compilation.

[1] The system can be downloaded at http://centria.di.fct.unl.pt/~cd/mweb/

The major contribution of the paper is the specification of the semantics of ERDF reasoning entirely in the MWeb framework, including alignment with RIF, support of RDF and RDFS entailment, as well extensions to the original ERDF semantics for dealing with closed classes and properties. These results complement the mapping of simple modular ERDF ontologies into MWeb rulebases defined in [7]. Thus reasoning on simple modular ERDF ontologies can be achieved through our MWeb implementation[2], and in particular supporting modular reasoning over RDF(S) ontologies.

The paper is organized as follows. In Section 2 we illustrate how simple modular ERDF ontologies are mapped into MWeb rulebases. Next Section 3, specifies the support of ERDF reasoning by MWeb logic rules instead of the formal model-theoretical presentation of [3]. The paper finishes with some conclusions.

2 The MWeb Embedding of ERDF Ontologies

Simple modular ERDF ontologies [3] allow the combination of knowledge in different ontologies. Specifically, a simple modular ERDF ontology (SMEO) is a set of simple r-ERDF ontologies. The language of simple r-ERDF ontologies allows the use of ordinary triples s.[p ->> o] and negated triples neg s.[p ->> o] in the ERDF graph, where s, p and o are respectively the subject, predicate and object of the statement. Additionally, it allows to construct programs using deductive rules to derive new (extended) triples by rules having bodies formed by combining the connectives naf (weak negation), neg (strong negation), and conjunction. Moreover, provides mechanisms to define modules of knowledge, which are described by an interface and formed by an ERDF graph and a program. Finally, it provides a means to query other rulebases via qualified literals of the form *Lit*@*URI* in rules. Details can be found in [3,7].

The MWeb framework requires for each rulebase (module of knowledge) the definition of an *interface* document and of the corresponding *rulebase* (*logic*) document. The MWeb interface is formed by a sequence of declarations. First, the name of the rulebase is stated via a rulebase declaration followed by an IRI. Optional base IRI and prefixes can be declared for simplifying writing of classes and property names, via a base and prefix declarations. Other interfaces may be recursively included via a special import declaration. This mechanism will be used to import the interfaces declaring the classes and properties defined by RIF, RDF, RDFS and ERDF. An optional vocabulary declaration can be used to list the vocabulary of the rulebase. Next, follow two blocks of declarations. The first block defines the predicates being defined in the MWeb rulebase, and correspond to a generalization of export declarations found in logic programming based languages. The second block correspond to generalization of import declarations. The interesting feature of the MWeb framework is that besides scope (i.e. internal, local, or global), different reasoning modes can be associated to predicates (i.e. definite, open, closed, or normal). This allows control of monotonicity of reasoning by the producer and consumer of the knowledge. In this work, all properties and classes are defined global (meaning that it can be defined in multiple rulebases) and normal (meaning that weak negation can be used). The semantics of all MWeb constructs can be found in [4] as well as additional motivation. In [7] it is defined the

[2] The MWeb system is available at http://centria.di.fct.unl.pt/~cd/mweb

```
:- rulebase '$Nam_O$'.
:- import('erdf.mw',interface).
:- vocabulary rdf:'_1',..., rdf:'_n'.
:- defines global normal class(mw:Vocabulary).
% For each class $c$ exported to $r_1,\dots r_n$
:- defines global normal class($c$) visible to '$r_1$',...,'$r_n$'.
% For each property $p$ exported to $r_1,\dots r_n$
:- defines global normal property($p$) visible to '$r_1$',...,'$r_n$'.
% For each class $c$ imported from $s_1,\dots s_m$
:- uses normal class($c$)from '$s_1$',...,'$s_m$'.
% For each property $p$ imported from $s_1,\dots s_m$
:- uses normal property($p$)from '$s_1$',...,'$s_m$'.
% Let $u_1,\dots,u_d$ be the r-ERDF ontologies on which $O$ depends
:- uses normal class(mw:Vocabulary)from'$u_1$',...,'$u_d$'.
```

Fig. 1. Simple Modular ERDF Ontologies Interface in MWeb

translation of simple modular ERDF ontologies into the MWeb framework, whose general interface document is illustrated in Figure 1.

The first declaration in Figure 1 identifies the rulebase, while the `import` directive includes in the interface the necessary declarations for supporting ERDF reasoning, namely the vocabularies of RDF, RDFS and ERDF. The `erdf.mw` interface and corresponding rulebase will be presented later on, and implements in MWeb itself the underlying semantics of modular ERDF ontologies, including RIF, RDFS and RDF combination. The next declaration lists a limited number of container membership properties to be included in the vocabulary, in the case that at least one occurs in the graph or in the program. However, the property `rdf:_1` is always declared by the `erdf` ontology. The vocabulary of the rulebase is collected in the pre-defined class `mw:Vocabulary` of the MWeb framework and made visible to the allowed potential importing rulebases. The rulebase vocabulary is used for providing the domain for (scoped) negation as failure, open and closed world assumptions.

The next group declares the exported classes and properties, via the `defines` declaration of MWeb. The important point is that all classes and properties are defined normal and global. This means that all classes and properties exported can be used and redefined (`global`), and that rules can use weak negation (`normal`) and thus are non-monotonic. The visibility list states where the class or property can be used.

The subsequent `uses` declarations correspond to the import part of the interface, and are used in normal mode (weak negation allowed). The meaning of the importing list is obvious. The last `use` declaration extends the vocabulary of the rulebase with the vocabulary of the rulebases in the dependencies (directly or indirectly used modules). Notice that the vocabulary used in the current interface and corresponding program documents is automatically included, and need not to be declared.

The translation of the logic document of an r-*ERDF ontology* is immediate since the syntax used to represent rules in ERDF and MWeb is almost identical. In general, the MWeb logic document can start with an optional `import` declaration which allows textual inclusion of the rules found in the imported document. Afterwards, the fact and rules can be stated. The exact translation of simple modular ERDF programs can be found in [7], which is not presented for lack of space. Several examples of concrete and full MWeb logic documents will appear in the rest of the paper.

3 Semantics of Modular ERDF Ontologies in MWeb

In this section we specify the semantics of ERDF entailment through MWeb rulebases. This will be achieved incrementally and hierarchically, by providing first the definition of the used RIF primitive predicates. Afterwards, the semantics of RDF will be defined and made compatible with RIF as prescribed in [12], and subsequently a rulebase will define RDFS. Finally, we take care of the features of ERDF entailment.

3.1 RIF Support

The supported Rule Interchange Format dialect implements fully the semantics of membership and subclass, frames, and equality (partially). Regarding connectives, the usual binary conjunction, as well as strong and weak negations are supported. In order to maintain compatibility with RIF and generality, all properties and classes are assumed to be global and normal, allowing for the use of weak negation in the bodies, and thus monotonicity cannot be guaranteed. The MWeb syntax recognizes frames of the form `?O.[?A1->>?V1,...,?An->>?Vn]` which internally are translated into a conjunction `'->'(?A1,?O,?V1),...,'->'(?An,?O,?Vn)` of the ternary predicate `'->'/3`. The other binary RIF predicates '=' (equality), '#' (member), and '##' (subclass) have the exact syntax of RIF, and for ease of presentation are infix operators. The semantics of these predicates are provided by the rulebase of Figure 2.

```
RIF interface (rif.mw)
:- rulebase 'http://www.w3.org/2007/rif'.
:- prefix rif='http://www.w3.org/2007/rif#'.
:- defines internal normal name:'='/2, name:'#'/2, name:'##'/2, name:'->'/3.
```

```
RIF rulebase (rif.rb)
% RIF member relation
?O # ?CL :- ?O # ?SCL, ?SCL ## ?CL.
neg ?O # ?SCL :- neg ?O # ?CL, ?SCL ## ?CL.
% RIF subclass relation
?C1 ## ?C3 :- ?C1 ## ?C2, ?C2 ## ?C3.
% RIF equality theory.
?T = ?T :- ?T # mw:Vocabulary.
?T1 = ?T2 :- ?T2 = ?T1.
?T1 = ?T3 :- ?T1 = ?T2, ?T2 = ?T3.
neg ?T1 = ?T2 :- neg ?T2 = ?T1.
neg ?T1 = ?T3 :- ?T1 = ?T2, neg ?T2 = ?T3.
% RIF frames obtained by equality reasoning.
?O.[?P ->> ?V] :- ?O1.[?P ->> ?V], ?O = ?O1.
?O.[?P ->> ?V] :- ?O.[?P1 ->> ?V], ?P = ?P1.
?O.[?P ->> ?V] :- ?O.[?P ->> ?V1], ?V = ?V1.
```

Fig. 2. MWeb Rulebase Implementing RIF Relations

The interface document just defines the RIF primitive predicates being implemented using predicate indicators (name/arity), and these are hidden with the internal keyword.

```
RDF interface (rdf.mw)
:- rulebase  'http://www.w3.org/1999/02/22-rdf-syntax-ns'.
:- prefix  rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'.
:- import( 'rif.mw', interface ).

:- vocabulary rdf:nil, rdf:'_1'.
:- defines internal normal class(rdf:Property), ...
:- defines internal normal property(rdf:type), ...
```

```
RDF rulebase (rdf.rb)
:- import( 'rif.rb', rulebase ).
% RDF compatibility with RIF makes # and rdf:type equivalent
 ?X # ?Y :- ?X.[ rdf:type ->> ?Y].
 ?X.[ rdf:type ->> ?Y] :- ?X # ?Y.
% RDF Entailment rule
 ?Z.[ rdf:type ->> rdf:Property] :- ?_.[?Z ->> ?_].

% RDF Axiomatic triples
 rdf:type.[rdf:type->>rdf:Property].        rdf:subject.[rdf:type->>rdf:Property].
 rdf:predicate.[rdf:type->>rdf:Property].  rdf:object.[rdf:type ->> rdf:Property].
 rdf:first.[rdf:type ->> rdf:Property].      rdf:rest.[rdf:type ->> rdf:Property].
 rdf:value.[rdf:type ->> rdf:Property].          rdf:nil.[rdf:type ->> rdf:List].

% Infinitely many membership properties. Uses side-effects to restrict.
?X.[rdf:type ->> rdf:Property] :-
 External(name:atom(?X),prolog), External(name:atom_concat(rdf:'_',?N,?X ),prolog),
 External(name:is_number_atom(?N),prolog).
```

Fig. 3. MWeb Rulebase Implementing RDF Entailment

Notice the use of the special prefix `name` which is associated with the empty string. Class inheritance is captured by the first rule in document '`rif.rb`'. Mark the need to have a rule for taking care of the case where it is known that something does not belong to the extension of some class (second rule). For subclass relation it is only required transitivity, which does not have a dual negative rule.

Equality rules implement reflexivity, commutativity and transitivity. Reflexivity is applied to the declared vocabulary collected in the pre-defined class `mw:Vocabulary`. The equality rules are restricted to frames, and cannot handle complex terms[3]. Notice that this is very similar to the rules of `owl:sameAs` in the OWL2 RL profile [11].

3.2 RDF Semantics

The semantics of the combination of RDF with rules is the one adopted by RIF and specified in [12]. An ordinary triple `s p o` is syntactically represented by the RIF frame `s.[p ->> o]`. RIF lists are not supported because would require introducing complex terms in the language.

The support of RDF entailment is immediate and can be found in Figure 3. All the classes and properties of the RDF vocabulary are declared in the interface document '`rdf.mw`'. Notice also the declaration of the prefix `rdf` to simplify writing of URIs using Compact URI notation (CURIEs). A class declaration `class(`$CURIE$`)` is short for `?_ #` $CURIE$, while the `property(`$CURIE$`)` is syntactic sugar for

[3] Full equality is not necessary for ERDF, whose terms are just URIs and literals.

```
RDFS interface (rdfs.mw)
:- rulebase 'http://www.w3.org/2000/01/rdf-schema'.
:- prefix rdfs='http://www.w3.org/2000/01/rdf-schema#'.
:- import( 'rdf.mw', interface ).

:- defines internal normal class(rdfs:Resource), class(rdfs:Literal),
   class(rdfs:Datatype), class(rdfs:Class), ...

:- defines internal normal property(rdfs:domain), property(rdfs:range),
   property(rdfs:subClassOf), property(rdfs:subPropertyOf), ...
```

```
RDFS rulebase (rdfs.rb)
:- import( 'rdf.rb', rulebase ).

% RDFS compatibility into RIF requires including ## into rdfs:subClassOf
?X.[rdfs:subClassOf ->> ?Y] :- ?X ## ?Y.

% Some of RDFS entailment rules
?Z.[ rdf:type ->> ?Y] :- ?X.[rdfs:domain ->> ?Y], ?Z.[?X ->> ?W].
?W.[ rdf:type ->> ?Y] :- ?X.[rdfs:range ->> ?Y], ?Z.[?X ->> ?W].
?Z.[ rdf:type ->> ?Y] :- ?X.[rdfs:subClassOf ->> ?Y], ?Z.[rdf:type ->> ?X].

?X.[rdfs:subClassOf ->> ?X] :- ?X.[rdf:type ->> rdfs:Class] .
?X.[rdfs:subClassOf ->> ?Z] :-
    ?X.[rdfs:subClassOf ->> ?Y], ?Y.[rdfs:subClassOf ->> ?Z].
    .
    .
    .
% other entailment and RDFS axiomatic triples follow
```

Fig. 4. MWeb Rulebase Implementing RDFS Entailment

the RIF frame?_.[$CURIE$->>?_] resulting in better looking interface documents. The ?_ occurrences represent anonymous variables.

By the recommendation governing RIF-RDF compatibility, the predicates '#'/2 and `rdf:type` should be made equivalent; this is achieved by the first rules in document '`rif.rb`'. The only rule necessary for RDF entailment states that any predicate of a triple must have type `rdf:Property`. Then, the axiomatic RDF triples are listed, concluding with the special treatment of RDF container membership properties.

The RDF container membership properties are handled by external calls to Prolog underlying system, since they are infinitely many (`rdf:_1`, `rdf:_2`, etc...), and their full inclusion would result in the generation of an infinite number of answers for some non-ground queries. The rule only fires if the subject of the triple is bound at query time with a ground atom.

3.3 RDFS Semantics

RDF Schema entailment is more complex to specify, but immediate. According to RIF-RDF compatibility every RIF subclass instance is also an `rdfs:subClassOf` instance (but not vice-versa). Afterwards, all the RDF schema inference rules and axiomatic triples are encoded; container membership properties are treated as in the implementation of RDF. Besides XML and plain literals, no other datatypes are handled.

A snippet of the MWeb implementation of RDFS entailment is present in Figure 4, and adopts the complete RDFS entailment rules of [14]. The interface is very similar to the one of RDF, adapted to the corresponding vocabulary.

3.4 ERDF Semantics

The Extended Resource Description Framework vocabulary introduces the notions of total and closed class, as well as total and closed property, and a mechanism to express complementary properties, whose interface file 'erdf.mw' is depicted below:

```
:- rulebase 'http://erdf.org'.
:- prefix erdf='http://erdf.org#'.
:- import( 'rdfs.mw', interface ).

:- defines internal normal class(erdf:TotalClass),
   class(erdf:PositivelyClosedClass), class(erdf:NegativelyClosedClass).

:- defines internal normal class(erdf:TotalProperty),
   class(erdf:PositivelyClosedProperty), class(erdf:NegativelyClosedProperty).

:- defines internal normal property(erdf:complementOf).
```

Totalness is enforced by declaring the class and property having `erdf:TotalClass` and `erdf:TotalProperty` type, respectively, corresponding to open world assumptions with respect to the declared vocabulary. Closed classes can be declared using type `erdf:PositivelyClosedClass` or `erdf:NegativelyClosedClass`, similarly `erdf:PositivelyClosedProperty`, and `erdf:NegativelyClosedProperty` can be used to declare closed properties. These correspond to closed world assumptions with respect to the declared vocabulary.

The semantics of the ERDF constructs is specified in the rulebase 'erdf.rb'. Obviously, the document 'erdf.rb' starts by importing the defining rules of RDFS, RDF and RIF rulebases with the initial declaration `:- import('rdfs.rb', rulebase).` The relationship to RIF primitive predicates is extended to negative extensions of the predicates by the following rules:

```
neg ?X # ?Y :- neg ?X.[ rdf:type ->> ?Y].
neg ?X.[rdf:type ->> ?Y] :- neg ?X # ?Y.
neg ?X.[rdfs:subClassOf ->> ?Y] :- neg ?X ## ?Y.
```

The next rule extends RDF entailment by assigning the type `rdf:Property` to properties which are used in negative triples:

```
?Z.[rdf:type->>rdf:Property] :- neg ?_.[?Z->>?_].
```

ERDF extends RDFS with rules for propagating "downwards" in the hierarchy negative class and negative property extensions:

```
neg ?Z.[rdf:type ->> ?X] :- ?X.[rdfs:subClassOf->>?Y], neg ?Z.[rdf:type->>?Y].
neg ?Z1.[?X ->> ?Z2] :- ?X.[rdfs:subPropertyOf->>?Y], neg ?Z1.[?Y->>?Z2].
```

The remaining rules take care of specificities of ERDF entailment itself. The semantics of total classes and properties are captured by the next rules encoding open-world assumptions. Since these rules require the use of negation as failure, we have to guarantee grounding of the free variables in order to avoid unsoundness of reasoning. The grounding of variables is made with respect to the rulebase declared vocabulary, and therefore is a scoped negation as failure:

```
neg ?Z.[rdf:type->>?X] :- ?X.[rdf:type ->> erdf:TotalClass],
   ?Z#mw:Vocabulary, naf ?Z.[rdf:type->>?X].
?Z.[rdf:type->>?X] :- ?X.[rdf:type ->> erdf:TotalClass],
   ?Z#mw:Vocabulary, naf neg ?Z.[rdf:type->>?X].
```

```
neg ?Z1.[?X->>?Z2] :- ?X.[rdf:type ->> erdf:TotalProperty],
    ?Z1#mw:Vocabulary, ?Z2#mw:Vocabulary, naf ?Z1.[?X->>?Z2].
?Z1.[?X->>?Z2] :- ?X.[rdf:type ->> erdf:TotalProperty],
    ?Z1#mw:Vocabulary, ?Z2#mw:Vocabulary, naf neg ?Z1.[?X->>?Z2].
```

The semantics of closed classes and properties are captured by using one of the rules in the definition for total ones. For instance, a positively closed class means that its positive instances are exhaustive, therefore all the remaining individuals in the vocabulary are known to not belonging to the class (this is captured in the first rule below). Negatively closed classes have a dual interpretation. The notion of positively and negatively closed properties are similar.

```
neg ?Z.[rdf:type->>?X] :- ?X.[rdf:type->>erdf:PositivelyClosedClass],
    ?Z#mw:Vocabulary, naf ?Z.[rdf:type->>?X].
?Z.[rdf:type->>?X] :- ?X.[rdf:type->>erdf:NegativelyClosedClass],
    ?Z#mw:Vocabulary, naf neg ?Z.[rdf:type->>?X].
neg ?Z1.[?X->>?Z2] :- ?X.[rdf:type->>erdf:PositivelyClosedProperty],
    ?Z1#mw:Vocabulary, ?Z2#mw:Vocabulary, naf ?Z1.[?X->>?Z2].
?Z1.[?X->>?Z2] :- ?X.[rdf:type->>erdf:NegativelyClosedProperty],
    ?Z1#mw:Vocabulary, ?Z2#mw:Vocabulary, naf neg ?Z1.[?X->>?Z2].
```

For legacy applications to be able to express negative triples in ordinary RDF graphs, the ERDF vocabulary includes a mechanism to state that properties are complementary with the property `erdf:complementOf`. The net effect is the exchange of the positive and negative instances of the complementary properties:

```
neg ?S.[?P->>?O] :- ?P.[erdf:complementOf->>?Q], ?S.[?Q->>?O].
neg ?S.[?P->>?O] :- ?Q.[erdf:complementOf->>?P], ?S.[?Q->>?O].
?S.[?P->>?O] :- ?P.[erdf:complementOf->>?Q], neg ?S.[?Q->>?O].
?S.[?P->>?O] :- ?Q.[erdf:complementOf->>?P], neg ?S.[?Q->>?O].
```

Finally, the axiomatic triples of ERDF are included which basically state that all classes in the ERDF vocabulary are subclasses of `rdfs:Class` and that the special property `erdf:complementOf` has domain and range `rdf:Property`. For lack of space these are not included here, but are trivial to state.

A direct translation into extended logic programming of the MWeb rulebases has been defined in [7], which uses a quad representation `'->'('m',p,s,o)` predicate to state that triple `s p o` is true at MWeb rulebase m. Briefly, a uses `class(`C`)` declaration in the interface of m generates, for each rulebase 'r_i' in its from list, the rule `'#'('m',?X,`C`) :- '#'('`r_i`',?X,`C`)`. A uses `property(`P`)` generates, for each rulebase 'r_i' in its from list, the rule `'->'('m',`P`,?S,?O):-'->'('`r_i`',`P`,?S,?O)`. Program rules are translated by introducing an extra first argument 'm' to all literals in the rule, except for qualified literals $L@o$ whose new (first) argument is 'o'. This translation has been shown sound and complete with respect to the simple modular ERDF ontologies semantics [3], and can be used for query answering of simple modular ERDF ontologies.

4 Discussion and Conclusions

This work provides a rule-based declarative specification of ERDF entailment in simple modular ERDF ontologies, via the embedding of simple modular ERDF ontologies into the MWeb framework filling in the details in [7]. The specification is based on a novel program transformation. It reports, to the best of our knowledge, the first complete approach combining for the first time RIF and RDFS semantics, and extends it to the case of graphs capable of expressing negative information and ontologies with open, closed world assumptions, and scoped negation as failure. The representation power is complemented with features for combining modularly ontologies in the Semantic Web.

A complete working system resorting to the MWeb implementation in XSB Prolog 3.2 has been developed, and is available for download with promising results. We performed a first comparison using the W3C's Wine ontology, determing CPU times for RDFS inference without equality reasoning (discarding loading and compile times). Briefly, Jena's (2.6.2) inbuilt RDFSReasoner was 2 times slower than our MWeb implementation, while Jena's Generic reasoner was 100 times slower. Euler Yap (Eye 3414) shown to be 4 times slower and CWM-1.2.1 was 100 times slower.

References

1. Alferes, J.J., Damásio, C.V., Pereira, L.M.: A Logic Programming System for Non-monotonic Reasoning. Journal of Automated Reasoning 14(1), 93–147 (1995)
2. Analyti, A., Antoniou, G., Damásio, C.V.: A Principled Framework for Modular Web Rule Bases and Its Semantics. In: Proc. of KR-2008, pp. 390–400. AAAI Press, Menlo Park (2008)
3. Analyti, A., Antoniou, G., Damásio, C.V.: A Formal Theory for Modular ERDF Ontologies. In: Polleres, A. (ed.) RR 2009. LNCS, vol. 5837, pp. 212–226. Springer, Heidelberg (2009)
4. Analyti, A., Antoniou, G., Damásio, C.V.: MWeb: a Principled Framework for Modular Web Rule Bases and its Semantics. Accepted in ACM Transactions on Computational Logic, TOCL (2010)
5. Analyti, A., Antoniou, G., Damásio, C.V., Wagner, G.: Extended RDF as a Semantic Foundation of Rule Markup Languages. Journal of Artificial Intelligence Research 32, 37–94 (2008)
6. Bao, J., Voutsadakis, G., Slutzki, G., Honavar, V.: Package-based description logics. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.) Modular Ontologies. LNCS, vol. 5445, pp. 349–371. Springer, Heidelberg (2009)
7. Damásio, C.V., Analyti, A., Antoniou, G.: Implementing Simple Modular ERDF ontologies. In: Proc. of 19th European Conference on Artificial Intelligence (to appear, 2010)
8. Ensan, F.: Formalizing Ontology Modularization through the Notion of Interfaces. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 74–82. Springer, Heidelberg (2008)
9. Gelfond, M., Lifschitz, V.: Logic programs with Classical Negation. In: 7th International Conference on Logic Programming (ICLP'90), pp. 579–597 (1990)
10. Grau, B.C., Parsia, B., Sirin, E.: Ontology integration using epsilon-connections. In: Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, pp. 293–320. Springer, Heidelberg (2009)
11. Reynolds, D. (ed.): OWL 2 RL in RIF W3C Working Group Note, June 22 (2010), http://www.w3.org/TR/rif-owl-rl/
12. de Bruijn, J. (ed.): RIF RDF and OWL Compatibility W3C Recommendation, June 22 (2010), http://www.w3.org/TR/rif-rdf-owl/
13. Serafini, L., Borgida, A., Tamilin, A.: Aspects of Distributed and Modular Ontology Reasoning. In: 19th Int. Joint Conf. on Artificial Intelligence, pp. 570–575 (2005)
14. ter Horst, H.J.: Completeness, Decidability and Complexity of Entailment for RDF Schema and a Semantic Extension Involving the OWL Vocabulary. Journal of Web Semantics 3(2-3), 79–115 (2005)

KWilt: A Semantic Patchwork for Flexible Access to Heterogeneous Knowledge

Klara Weiand[1], Steffen Hausmann[1], Tim Furche[1,2], and François Bry[1]

[1] Institute for Informatics, University of Munich,
Oettingenstraße 67, D-80538 München, Germany
http://www.pms.ifi.lmu.de/

[2] Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford, OX1 3QD, England
http://web.comlab.ox.ac.uk/people/Tim.Furche/

Abstract. Semantic wikis and other modern knowledge management systems deviate from traditional knowledge bases in that information ranges from unstructured (wiki pages) over semi-formal (tags) to formal (RDF or OWL) and is produced by users with varying levels of expertise. KWQL is a query language for semantic wikis that scales with a user's level of expertise by combining ideas from keyword query languages with aspects of formal query languages such as SPARQL. In this paper, we discuss KWQL's implementation KWilt: It uses, for each data format and query type, technology tailored to that setting and combines, in a patchwork fashion, information retrieval, structure matching and constraint evaluation tools with only lightweight "glue". We show that it is possible to efficiently recognize KWQL queries that can be evaluated using only information retrieval or information retrieval and structure matching. This allows KWilt to evaluate basic queries at almost the speed of the underlying search engine, yet also provides all the power of full first-order queries, where needed. Moreover, adding new data formats or abilities is easier than in a monolithic system.

1 Introduction

To accommodate all users, modern knowledge management systems such as semantic wikis must deal with both unstructured (textual or multi-modal) information as well as structured data carrying varying degrees of semantics: hierarchical data for document and simple classification structures, social classifications in form of tag networks, formal ontologies in RDF or OWL. Expert users in such systems can define semantically rich, automated analysis or derivation tasks. However, the vast number of users has little understanding of formal knowledge representation, produces unstructured information with lightweight semantic annotations such as free-form tags, and interacts with the system through simple but imprecise keyword queries.

From these observations, we derive two properties that characterize successful modern knowledge management systems:

(1) "Interfaces must be adaptable and flexible": Interfaces should scale with user experience: For novice users, simple, but imprecise queries are useful for satisfying their information needs; for expert users precise, but necessarily fairly complex queries that enable automated action and derivation are required. Interfaces should also be able to adapt to different types of knowledge in a system, providing a consistent interface.

(2) "Patchwork knowledge management": Due to the growth in data size and formats, knowledge management systems face a dual challenge: Users expect high performance for (at least basic) queries regardless of the data scale, as in Web search engines. On the other hand, knowledge management systems must be able to adapt quickly to additional knowledge sources, providing scalable yet sufficiently expressive interfaces to query and process such data.

In this paper, we present a patchwork approach to knowledge management using the query language KWQL and its implementation, KWilt. We choose the setting of a semantic wiki, KiWi, as it exemplifies many of the challenges outlined above.

KWQL: Scale with User Experience. To illustrate how KWQL provides a consistent interface that easily adapts to different levels of user experience, consider the following scenario: *"In a wiki describing KiWi, we would like to find all wiki pages that describe (knowledge management) systems that have influenced the development of KiWi."*

In a conventional knowledge management system, we would expect a formal relation (e.g., `wk:influences`) that represents the very intent of our query. Indeed, given an RDF representation of such a query we can query such a relation in KWQL as (assuming `wk:KiWi` represents the KiWi system):

```
ci(rdf(predicate:'wk:influences' object:'wk:KiWi'))
```

However, in most cases this relation is *not* present explicitly. Even if it is, users are often not able to express their intent in such a formal manner.

Accustomed to Web search engines, novice users might start with a keyword query that returns all resources (or **c**ontent **i**tems) containing "KiWi": `KiWi`

Obviously, such a query is too unspecific to capture the above query intent and, in fact, may omit a number of systems, that are described without reference to KiWi, but that are referenced from the description of KiWi.

Thus, we might refine the query to return such referenced resources, i.e., resources that are the target of a link originating from a wiki page containing "KiWi": `$u @ ci(KiWi link(target:ci(URI:$u))`

The `$u @` of the query ensures, that not the content item that points to the page, but rather the page that is pointed at, is returned by the query. However, that query is not specific enough, as it returns also, e.g., technologies used in KiWi. We know that KiWi is a semantic wiki and might be tempted to amend that query to return only resources that are also semantic wikis:

```
$u @ ci(KiWi link(target:ci(URI:$u tag(name:'semantic wiki')))
```

But there might well be other systems that have a significant influence on KiWi. To also capture them, we choose the query:

```
$u @ ci(KiWi tag(name:$t) link(target:ci(URI:$u tag(name:$t)))
```

It returns all resources that are tagged the same as a wiki page containing "KiWi" that also links to the returned resource. This way we likely capture resources with similar characteristics as KiWi that are also mentioned in its description.

To summarize, KWQL's main contributions over existing query languages and similar interfaces for knowledge management systems are:

1. KWQL provides a **consistent interface** for access to the wide range of knowledge present in the semantic wiki KiWi.
2. KWQL is designed to **scale with the user experience**: Queries can take the form of bags of keywords, but also be extended with increasingly more precise constraints on the structure, tags, and formal annotations of wiki pages.
3. KWQL is well **integrated into KiWi**, it covers all aspects of its data model and is used in KiWi's rule language.

KWilt: Patchwork Knowledge Management. Previous approaches have often tried to engineer a knowledge information systems for such diverse information and user needs from the start. In constrast, KWilt, KWQL's implementation in KiWi, uses a "patchwork" approach, combining performant and mature technologies where available. For example, KWilt uses a scalable and well established information retrieval engine (Solr) to evaluate keyword queries. In fact, KWilt tries to evaluate as large a fragment of any KWQL query in the information retrieval engine as possible. If necessary, the results are further refined by (1) checking any structural constraints of the query and (2) finally enforcing all remaining first-order constraints, e.g., from multiple variable occurrences.

KWilt's patchwork approach has three main advantages:

1. Many queries can be evaluated at the speed of search engines, yet all the power of first-order logic is available if needed, as detailed in Section 4: The three steps use **increasingly more expressive, but also less scalable** technologies. Thus even for queries that involve full first-order constraints, we can, in most cases, substantially reduce the number of candidates in the information retrieval engine and by enforcing structural constraints before evaluating the first-order constraints.
2. Each part is implemented using **proven technologies** and algorithms with minimal "glue" between the employed tools, see Section 2.
3. The **separation makes it easy to adapt** each of the parts, e.g., to reflect additional data sources. E.g., if KiWi would introduce data with different structural properties, e.g., strictly hierarchical taxonomies in addition to RDF ontologies only the part of KWilt that evaluates structural constraints needs to be modified. Similarly, if KWQL would introduce other content primitives other than keywords (e.g., for image retrieval), only the first (retrieval) part of KWilt would be affected.

2 KWilt: Architecture and Evaluation Phases

Evaluating KWQL queries is a challenging task that cannot be accomplished by existing query engines for (semantic) wikis. For instance, the query

```
ci(KiWi tag(name:$t) link(target:ci(tag(name:$t))))
```

combines content and structural elements with variables to find content items that a wiki page containing *"KiWi"* links to and that have at least one tag in common with their linking page.

Despite the unique combination of features found in KWQL, KWilt does not try to "reinvent the wheel". In particular, we have chosen not to build a new index structure capable of combining all these aspects in a single index access, as this approach has several drawbacks. First and foremost, the rich data model would require a fairly complex index structure that can support content and structure queries, fast access to hierarchical data and link graphs, RDF graph navigation as well as navigation over (simpler, but less regular) containment and link relations. Moreover, it is quite likely that the data model evolves over time with new kinds of data or different representation formats introduced, in particular in the field of social semantic media. Using a complex index structure which is carefully adapted for a certain data model makes it hard or even infeasible to react to these potential changes.

Instead, we used an *patchwork*, or integration, approach to combine off-the-shelf state-of-the-art tools in a single framework. For that, the evaluation is split into **three different evaluation phases** which are dedicated to certain aspects of the query. Each step makes use of a tool which is particularly suitable for evaluating the query constraints covered by that aspect of the entire evaluation, e.g., for keyword queries we use a traditional search engine. Thus, efficient and mature algorithms form the basis of our framework while the framework itself remains flexible with lightweight "glue" to combine the evaluation phases.

Evaluating keyword queries: Most KWQL queries, in particular by novice users, mostly or only regard the content of the pages. Therefore, the first evaluation phase regards the keyword parts of a query in order to evaluate them in an early phase of the evaluation with as little overhead as possible. In particular, if all constraints of the query can be validated in this phase, the two subsequent phases can be skipped.

The information retrieval engine Solr provides a highly optimized inverted list index structure to carry out keyword queries on a set of documents. Each document consists of an arbitrary number of named fields which are most commonly used to store the text of a document and its meta data. In order to benefit from Solr, the content of the wiki needs to be stored in this index, i.e., *all resources including their dependencies need to be translated to flat Solr documents.*

In order to use Solr for the evaluation of KWQL queries, the meta data of wiki pages and further the meta data of its tags, fragments and links are stored in a Solr document. The main principle of the translation is to *materialize joins* between content items and the directly connected resources (tags, fragments, links) that are commonly queried together in the same query. These materialized joins are then stored in the fields of the document representing the content item. Thus, queries regarding the meta data of content items and even the meta data of its tags, fragments and links can be directly answered with Solr. However, the transformation of the resources connected to a content item to fields in the Solr index is lossy, since the value of multiple resources is stored in a single field. Thus, if multiple properties of a resource are queried, it cannot be guaranteed that hits in the index belong to the same resource. Therefore is necessary for certain kinds of queries to validate the generated result set of Solr.

To keep the index small only *dependencies to flat resources* (that can not be further nested) are materialized, which omits in particular nesting and linking of content items. Therefore, only queries that access content items together with their content, meta-data and directly related flat resources can be evaluated entirely in Solr. As soon as nesting and linking of content items comes into play, however, we use Solr only to generate a set of candidates which match those parts of the query for which all necessary information stored in the Solr index.

In order to evaluate a KWQL query through Solr, a portion of the KWQL query (that can be evaluated by Solr) is converted to the query language of Solr. Information which is not covered by the materialized joins and variables are either disregarded or at least converted to an existential quantification in order to reduce the number of false positives.

Evaluating structural constraints: The second phase takes the structural parts of a query into account. All resources are represented as common objects in the KiWi system and their dependencies are modeled by references between the interrelated objects. The objects are persisted using a common relational database in combination with an object relational mapping.

In the current prototype, we validate the structural properties of a query for each candidate item individually. That means, nested resources (tags, fragments, links and contained content items) which are specified in the query are considered by traversing the references of the currently investigated object.

We choose this approach, as structural constraints are often validated fairly quickly and far less selective than the keyword portions of KWQL queries. However, for future work we envision an extension of KWilt that improves on the current implementation in two aspects: (a) It estimates whether the structural part is selective enough to warrant its execution without considering the candidates from the previous phase, followed by a join between the candidate sets from the two phases. (b) If structural constraints become more complex, specialized evaluation engines for hierarchical (XML-style) data, e.g, a high-performance XPath engine, for link data, e.g., various graph reachability indices, and for RDF data might be advantages.

In addition to the verification of the structural constrains, the structural dependencies of the contributing resources and the required values of their qualifiers are stored in relations which are needed during the last evaluation phase. For instance the titles of content items are stored in the relation R_{title} which is therefore a set of tuples of identifiers and strings.

Evaluating first-order constraints over wiki resources: In the final evaluation phase, first-order constraints over wiki resources are considered, as induced by the KWQL variables (and some advanced features of KWQL such as injectivity, that are not further discussed here).

Following constraint programming notation, we consider a first-order constraint a formula over logical relation on several variables. In order to use these constraints to express a KWQL query, every expression *of a query* that is involved in constraints not yet fully validated is represented by some variables. These variables are then connected using relations which reflect the structural constraints between the resources from the query and their meta data. These relations are constructed during the prior evaluation phase since all required values and dependencies of the resources are regarded in this phase anyhow.

For instance, to express that a content item has a certain title, the relation R_{title} is used: $(C, KiWi) \in R_{\text{title}}$. This constraint causes the variable C to be bound only to identifiers of content items with the title *"KiWi"*. Likewise, the relation R_{tag} is used to specify that a content item has a tag: $(C, T) \in R_{\text{tag}}$.

For each KWQL variable of the query, a new first-order variable is generated that can be used in the structural constraints. For instance, the query

```
ci(title:$t tag(name:$t))
```

can be represented as: $(C, \$t) \in R_{\text{title}} \wedge (C, T) \in R_{\text{tag}} \wedge (T, \$t) \in R_{\text{name}}$.

Thus the relations are used to connect the formal representation of the query and the candidate matches. In case of the example, the constraints ensure that the content item's title is equal to the name of one of its tags. The first-order constraints are evaluated using the constraint solver `choco`.

Any content item that fulfills the constraints validated in all three phases is a match for the entire query. In fact, since we only feed candidate matches from the prior phase to each subsequent phase, the content item (identifiers) returned by `choco` immediately give us the KWQL answers.

3 Skipping Evaluation Phases: KWQL's Sublanguages

The evaluation of a general KWQL query in KWilt is performed in three phases as described in the previous section. However, not all evaluation phases are required for every KWQL query. In the following, we give a characterization of KWQL queries that can be evaluated using only the first phase (and skipping the remaining ones), or only the first and second.

Keyword KWQL or KWQL$_K$ is the restriction of KWQL to mostly flat queries where *resource terms* may not occur nested inside other resource terms and *structure terms* are not allowed at all.

Since tags and fragments itself can not be nested more than one level, we can also materialize all tags and fragments for each content item. However, in contrast to (string-valued) qualifiers a content item can have multiple tags or fragments. To allow evaluation with a information retrieval engine such as Solr, we have to ensure that multiple tag or fragment expressions always match with different tags or fragments of the surrounding content item. This avoids that we have to enforce the injectivity of these items in a later evaluation phase.

To ensure this, we allow tag and fragment queries but disallow 1. two keyword queries as siblings expressions in tag or fragment queries and 2. two tag or fragment queries as sibling expressions

KWQL$_K$ expressions can be evaluated entirely by the information retrieval engine (here: Solr). This is obvious for keywords. String-valued properties, tags, and fragments and qualifiers are materialized in Solr together with their resources (as specific fields) and thus can be queried through Solr as well.

Tree-shaped KWQL or KWQL$_T$ allows only queries corresponding to tree-shaped constraints. Thus, no multiple occurrences of the same variable, and no potentially overlapping expression siblings.

Intuitively, two expressions are called *overlapping* if there is a KWQL node in any document that is matched by both expressions. E.g., `ci(tag(Java))` and `Java` are overlapping since both match content items. Unfortunately, this definition of *overlapping* does not lead to an efficient syntactic condition, as it is easy to see that containment of KWQL queries is special case of overlapping. Further, containment of KWQL queries is NP-hard by reduction from containment of conjunctive queries.

Therefore, we define an equivalence relation on expressions, called *potential overlap*, as a conservative approximation of overlapping. It holds between two expressions if they have the same return type in the KWQL semantics or if the return type of one is a subset of that of the other one. E.g., `desc:ci(Lucene)` and `child:ci(Java)` potentially overlapping, but `target:ci(Java)` does not overlap with either. This is only an approximation. For instance, `child:ci( URI:a)` and `child:ci(URI:b)` potentially overlap, though each content-item has a unique URI and thus the two expressions never actually overlap.

KWQL$_T$ expressions can be evaluated by using only Solr and checking the remaining structural conditions in the second evaluation phase. Full first-order constraints are not needed and the third (`choco`) phase can be skipped.

Proposition 1. *Given an arbitrary KWQL query, we can decide in linear time and space in the size of the query if that query is a KWQL$_K$ query and in quadratic time if it is a KWQL$_T$ query.*

Proof. From the definitions of KWQL$_K$ and KWQL$_T$ it is easy to see that testing membership of a general KWQL expression can be done by a single traversal of the expression tree. In the case of KWQL$_T$ we also have to test each (of the potentially quadratic) pairs of siblings for overlap and storing already visited variables.

4 Evaluating a KWQL Query in KWilt

Experiments were performed to analyze the evaluation times for queries of all three types of queries in a prototype implementation of KWilt. The KiWi system was run on a 2.66GHz Quad-Core iMac and filled with a data set of 431

Table 1. Evaluation times for various KWQL queries

Query	evaluation time
`KiWi`	34 ms
`KiWi tag(name:$t)`	36 ms
`ci(text:KWQL title:KiWi)`	29 ms
`ci(KiWi tag(name:KiWi) link(target:ci(URI:$u tag(name:KiWi))))`	416 ms
`ci(KiWi tag(name:$t) link(target:ci(URI:$u tag(name:$t))))`	580 ms
`ci(tag(name:$t) link(target:ci(URI:$u tag(name:$t))))`	796 ms

content items describing the KiWi project and KWQL. The reasoning and information retrieval modules of KiWi were deactivated to constrict the amount of background activity in the system. Every query was evaluated fifty times and the average was taken. The resulting evaluation times are given in table 1.

During evaluation, all queries are first parsed and then (partially) translated to the query language of Solr. For example, the query from the introduction

```
$u @ ci(KiWi tag(name:$t) link(target:ci(URI:$u tag(name:$t)))
```

is translated to the following Solr query:

```
type:ci AND (title:KiWi OR text:KiWi OR ...) AND tags:[* TO *]
```

Here, not only the keyword *KiWi* is included in the query but also the query for the tag, but since Solr does not support variables, the query for the tag is an existence constraint (indicated by `[* TO *]` as value of the tags qualifier).

The first three queries in the table can be fully captured with Solr and no further steps are needed. This is reflected in their low evaluation times. The other three queries contain constraints for validation in the subsequent phases.

In the next evaluation phase, the structural properties of the content items are validated against the query constraints. In order to gain full access to all properties of the content item, not just the simplified version stored in the Solr index, but the full representation is retrieved from the KiWi database.

This step is required to evaluate the "target" qualifiers in the remaining three queries. After the second evaluation phase, the fourth query has been fully evaluated. Its evaluation time is higher than that of the queries without structural constraints, but lower than the evaluation times of the remaining two queries which are further evaluated in the third evaluation phase[1].

In the last evaluation phase, all valid variable bindings are determined. Therefore, the query, or rather the still unverified parts of the query, are expressedin

[1] The current prototype *always* performs preparation for phase three in phase two.

a way suitable for the constraint solver `choco`. To this end, the relations containing the information about the resources are connected by variables.

$$\begin{aligned}(C_1, T_1) \in R_{\text{tag}} \wedge (T_1, \$t) \in R_{\text{name}} \wedge (C_1, L) \in R_{\text{link}} \wedge (L, C_2) \in R_{\text{target}} \\ \wedge (C_2, \$u) \in R_{\text{URI}} \wedge (C_2, T_2) \in R_{\text{tag}} \wedge (T_2, \$t) \in R_{\text{name}}\end{aligned}$$

The constraint solver then tries to determine bindings for all variables which satisfy the given constraint. The variable `$t` ensures that both tags of the query have the same name, whereas `$u` is used to obtain the URI of the content item that is pointed at, which will be returned as an answer to the query.

If the constraint solver does not succeed in finding a valid binding for the variables the content item is dropped from the candidate set, since it does not fulfill the constraints and therefore does not match the query.

5 Conclusion

To summarize, KWQL and KWilt together address two of the main challenges raised by the "democratization" of knowledge management driven by social technologies such as semantic wikis. KWQL provides a consistent interface for accessing knowledge in the semantic wiki KiWi. It addresses both the needs of novice users accustomed to simple, yet imprecise keyword interfaces, and of expert users that aim to write precise queries for automated processing.

KWilt implements KWQL by combining existing, proven technologies. This patchwork query engine allows us to quickly adapt to changes in the data formats and querying capabilities required by KiWi and its users. On the other hand, it also provides a stable, performant platform for search in a Wiki. Basic, keyword queries can be evaluated nearly at the speed of the underlying search engine and more complex queries can benefit from the fast filter phase.

The prototype of KWilt is already integrated in the current KiWi pre-release. First results illustrating KWilt's performance on the different types of queries were presented in this paper, demonstrating the effectiveness of our approach.

Acknowledgment. The research leading to these results is part of the project *"KiWi—Knowledge in a Wiki"* and has received funding from the European Community's 7th Framework Programme (FP7/2007-2013) under grant agreement No. 211932.

References

1. Chen, Y., Aberer, K.: Combining pat-trees and signature files for query evaluation in document databases. In: Bench-Capon, T.J.M., Soda, G., Tjoa, A.M. (eds.) DEXA 1999. LNCS, vol. 1677, pp. 473–484. Springer, Heidelberg (1999)
2. Goldman, R., Widom, J.: Dataguides: Enabling query formulation and optimization in semistructured databases. In: VLDB (1997)
3. Kaushik, R., Krishnamurthy, R., Naughton, J.F., Ramakrishnan, R.: On the integration of structure indexes and inverted lists. In: SIGMOD (2004)

4. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: a database management system for semistructured data. SIGMOD Record 26(3) (1997)
5. Milo, T., Suciu, D.: Index structures for path expressions. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 277–295. Springer, Heidelberg (1998)
6. Shimizu, T., Yoshikawa, M.: Full-text and structural indexing of XML documents on B^+-tree. IEICE Trans. on Information and Systems 89-D(1) (2006)
7. Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., Pan, Y.: Semplore: A scalable IR approach to search the web of data. Web Semantics: Science, Services and Agents on the World Wide Web, 7(3) (2009)
8. Wang, H., Park, S., Fan, W., Yu, P.S.: ViST: a dynamic index method for querying XML data by tree structures. In: SIGMOD (2003)
9. Weigel, F., Meuss, H., Bry, F., Schulz, K.U.: Content-aware DataGuides: Interleaving IR and DB indexing techniques for efficient retrieval of textual XML data. In: McDonald, S., Tait, J.I. (eds.) ECIR 2004. LNCS, vol. 2997, pp. 378–393. Springer, Heidelberg (2004)
10. Zou, Q., Liu, S., Chu, W.W.: Ctree: a compact tree for indexing XML data. In: WIDM, pp. 39–46 (2004)

Composition of Semantic Web Services in a Constructive Description Logic

Loris Bozzato and Mauro Ferrari

Dipartimento di Informatica e Comunicazione
Università degli Studi dell'Insubria
Via Mazzini 5, 21100, Varese, Italy

Abstract. We formalize the problem of service composition in the framework of a constructive description logic. We propose a declarative service specification language and a calculus for service composition.

1 Introduction

Semantic Web services are descriptions of the capabilities and the structure of services in the languages of the semantic Web. The current proposals for the representation of semantic Web services, as OWL-S [5], view services as processes with pre- and post- conditions and effects. The representation by pre- and post-conditions describe the requirements and output of a service that is useful to retrieve the service; the representation of the process associated with a service describe the interaction with other given services.

One of the main problems in the context of Web services is their composition. The problem can be stated as follows: given a composition goal, represented as a service with pre- and post- conditions, compose the available services so to satisfy the goal. Obviously in this context the challenge is to provide tools to support the definition of the composite service or, at best, to automatize the entire composition process. In this paper we discuss the problem of service composition in the context of the constructive description logic $\mathcal{BCDL}$ [4].

The main goal of this paper is to show how constructive DLs provide a natural framework for studying the problem of services composition. A detailed discussion and exemplification of the result of this paper is given in [2]. In this paper we simply detail manual composition of services, but by implementing our composition calculus we would obtain a method for automatic composition. Towards this, we need an implementation of $\mathcal{BCDL}_0$ and mappings from standard specification languages. For automatic composition, we plan to study the properties of our calculus and $\mathcal{BCDL}_0$, also comparing with $\mathcal{BCDL}$ and $\mathcal{KALC}$, a constructive DL for which a terminating tableaux procedure has been presented in [3], and with software synthesis and action formalisms.

2 Service Composition in $\mathcal{BCDL}_0$

The underlying logic. The language $\mathcal{L}$ of $\mathcal{BCDL}_0$ is same of $\mathcal{ALC}$ starting from the sets `NR` of *role names*, `NC` of *concept names* and $\mathcal{N}$ of *individual names*. Concepts are defined as in $\mathcal{ALC}$ from the constructors $\sqcap$, $\sqcup$, $\neg$, $\forall$ and $\exists$. A formula K over $\mathcal{L}$ is either $\bot$, a *role formula* $(c,d):R$, a *concept formula* $c:C$ or a subsumption $A \sqsubseteq C$, where $c,d \in \mathcal{N}$, $R \in \mathtt{NR}$, $A \in \mathtt{NC}$ and C is any concept. A theory T consists of a *TBox* and an *ABox*. A TBox is an acyclic finite set of formulas of the form $A \sqsubseteq C$. An ABox is a finite set of concept and role assertions.

Models $\mathcal{M} = (\mathcal{D}^{\mathcal{M}}, \cdot^{\mathcal{M}})$ for $\mathcal{L}$ coincide with the usual $\mathcal{ALC}$ models [1]. A formula K holds in $\mathcal{M}$, and we write $\mathcal{M} \models K$ if: $K \neq \bot$; $K = (c,d):R$ and $(c^{\mathcal{M}}, d^{\mathcal{M}}) \in R^{\mathcal{M}}$; $K = c:C$ and $c^{\mathcal{M}} \in C^{\mathcal{M}}$; $K = A \sqsubseteq C$ and $A^{\mathcal{M}} \subseteq C^{\mathcal{M}}$.

In $\mathcal{BCDL}_0$ a formula K is interpreted w.r.t an *information term* [4], that is a structured object α that provides a justification for the validity of K in $\mathcal{M}$. Given a formula K and a model $\mathcal{M}$ for $\mathcal{L}$, the set of information terms $\text{IT}(K)$ and the *realizability relation* $\mathcal{M} \rhd \langle\alpha\rangle\, K$ are defined as follows:[1]

- For K simple formula[2], $\text{IT}(K) = \{\mathtt{tt}\}$ and $\mathcal{M} \rhd \langle\mathtt{tt}\rangle\, K$ iff $\mathcal{M} \models K$.
- $\text{IT}(c : C_1 \sqcap C_2) = \text{IT}(c : C_1) \times \text{IT}(c : C_2)$ and $\mathcal{M} \rhd \langle(\alpha,\beta)\rangle\, c : C_1 \sqcap C_2$ iff $\mathcal{M} \rhd \langle\alpha\rangle\, c : C_1$ and $\mathcal{M} \rhd \langle\beta\rangle\, c : C_2$.
- $\text{IT}(c : C_1 \sqcup C_2) = \text{IT}(c : C_1) \uplus \text{IT}(c : C_2)$ and $\mathcal{M} \rhd \langle(k,\alpha)\rangle\, c : C_1 \sqcup C_2$ iff $\mathcal{M} \rhd \langle\alpha\rangle\, c : C_k$.
- $\text{IT}(c : \exists R.C) = \mathcal{N} \times \bigcup_{d\in\mathcal{N}} \text{IT}(d : C)$ and $\mathcal{M} \rhd \langle(d,\alpha)\rangle\, c : \exists R.C$ iff $\mathcal{M} \models (c,d) : R$ and $\mathcal{M} \rhd \langle\alpha\rangle\, d : C$.
- $\text{IT}(c : \forall R.C) = (\bigcup_{d\in\mathcal{N}} \text{IT}(d : C))^{\mathcal{N}}$ and $\mathcal{M} \rhd \langle\phi\rangle\, c : \forall R.C$ iff $\mathcal{M} \models c : \forall R.C$ and, for every $d \in \mathcal{N}$, if $\mathcal{M} \models (c,d) : R$ then $\mathcal{M} \rhd \langle\phi(d)\rangle\, d : C$
- $\text{IT}(A \sqsubseteq C) = (\bigcup_{d\in\mathcal{N}} \text{IT}(d : C))^{\mathcal{N}}$ and $\mathcal{M} \rhd \langle\phi\rangle\, A \sqsubseteq C$ iff, $\mathcal{M} \models A \sqsubseteq C$ and, for every $d \in \mathcal{N}$, if $\mathcal{M} \models d : A$ then $\mathcal{M} \rhd \langle\phi(d)\rangle\, d : C$

An information term for a set Γ of formulas is a function η mapping every $K \in \Gamma$ in an element of $\text{IT}(K)$. $\mathcal{M} \rhd \langle\eta\rangle\, \Gamma$ iff $\mathcal{M} \rhd \langle\eta(K)\rangle\, K$ for every $K \in \Gamma$.

$\mathcal{BCDL}_0$ is a subsystem of $\mathcal{BCDL}$ [4]. A sound and complete natural deduction calculus $\mathcal{ND}$ for $\mathcal{BCDL}$ is presented in [4], where it is also shown how $\mathcal{ND}$ supports the proofs-as-programs paradigms. In particular, given a proof π of a formula K from a set of formulas Γ we can extract from π an operator Φ_π associating with every $\eta \in \text{IT}(\Gamma)$ a $\alpha \in \text{IT}(K)$ such that $\mathcal{M} \rhd \langle\eta\rangle\, \Gamma$ implies $\mathcal{M} \rhd \langle\alpha\rangle\, K$. This result also holds for $\mathcal{BCDL}_0$ and it is the main result needed in the following.

Service definition. A *service specification* has the form $\mathsf{s}(x) :: P \Rightarrow Q$ where: s is a label identifying the service; x is the input parameter of the service (ranging over $\mathcal{N}$); P and Q are concepts. P is called the service *pre-condition*, denoted with $\text{Pre}(\mathsf{s})$, and Q the service *post-condition*, denoted with $\text{Post}(\mathsf{s})$. A *service implementation* is a function $\Phi_s : \bigcup_{t\in\mathcal{N}} \text{IT}(t : P) \to \bigcup_{t\in\mathcal{N}} \text{IT}(t : Q)$. We call *service definition* the pair $\mathcal{S} = (\mathsf{s}(x) :: P \Rightarrow Q, \Phi_s)$. Given a model $\mathcal{M}$, Φ_s *uniformly solves* $\mathsf{s}(x) :: P \Rightarrow Q$ in $\mathcal{M}$ iff, for every $c \in \mathcal{N}$ and every $\alpha \in \text{IT}(c : P)$ if $\mathcal{M} \rhd \langle\alpha\rangle\, c : P$ then $\mathcal{M} \rhd \langle\Phi_s(\alpha)\rangle\, c : Q$.

[1] Given two sets A and B, $A \uplus B$ denotes their disjoint union and B^A denotes the set of functions from A to B.

[2] *Simple formulas* are $\bot$, role formulas and concept formulas $c : C$ with C either an atomic or a negated concept.

$$\boxed{\dfrac{\mathsf{s}(x) :: A \Rightarrow B}{\begin{array}{c}\mathsf{s}_1(x) :: A_1 \Rightarrow B_1\\ \dots\\ \mathsf{s}_n(x) :: A_n \Rightarrow B_n\end{array}}\,\textsc{and}} \qquad \begin{array}{l}\text{AC} \begin{cases}(a_k)\ \mathbf{T}, x : A \vdash x : A_k,\ k = 1, \dots, n\\ (b)\ \ \mathbf{T}, x : B_1 \sqcap \dots \sqcap B_n \vdash x : B\end{cases}\\ \text{CI}\ \ \Phi_s(\alpha) = \Phi_b(\Phi_{s_1}(\Phi_{a_1}(\alpha)), \dots, \Phi_{s_n}(\Phi_{a_n}(\alpha)))\end{array}$$

$$\boxed{\dfrac{\mathsf{s}(x) :: A \Rightarrow B}{\begin{array}{c}\mathsf{s}_1(x) :: A_1 \Rightarrow B_1\\ \dots\\ \mathsf{s}_n(x) :: A_n \Rightarrow B_n\end{array}}\,\textsc{case}} \qquad \begin{array}{l}\text{AC} \begin{cases}(a)\ \ \mathbf{T}, x : A \vdash x : A_1 \sqcup \dots \sqcup A_n\\ (b_k)\ \mathbf{T}, x : B_k \vdash x : B,\ k = 1, \dots, n\end{cases}\\ \text{CI}\ \ \Phi_s(\alpha) = \Phi_{b_k}(\Phi_{s_k}(\alpha_k)) \text{ with } (k, \alpha_k) = \Phi_a(\alpha)\end{array}$$

$$\boxed{\dfrac{\mathsf{s}(x) :: A \Rightarrow B}{\begin{array}{c}\mathsf{s}_1(x) :: A_1 \Rightarrow B_1\\ \dots\\ \mathsf{s}_n(x) :: A_n \Rightarrow B_n\end{array}}\,\textsc{seq}} \qquad \begin{array}{l}\text{AC} \begin{cases}(b_1)\ \mathbf{T}, x : A \vdash x : A_1\\ (b_k)\ \mathbf{T}, x : B_{k-1} \vdash x : A_k,\ k = 2, \dots, n\\ (c)\ \ \mathbf{T}, x : B_n \vdash x : B\end{cases}\\ \text{CI}\ \ \Phi_s(\alpha) \ = \ \Phi_c(\,\Phi_{s_n} \cdot \Phi_{b_n} \cdot \ \dots \ \cdot \Phi_{s_1} \cdot \Phi_{b_1}(\alpha)\,)\end{array}$$

$$\boxed{\mathsf{s}(x) :: A \Rightarrow B \quad \text{AX}} \qquad \text{AC}\ \ (a)\ \mathbf{T}, x : A \vdash x : B \qquad \text{CI}\ \ \Phi_s(\alpha) \ = \ \Phi_a(\alpha)$$

$$\boxed{\mathsf{s}(x) :: A \Rightarrow B \quad \text{ENV}} \qquad \text{with } (\mathsf{s}, \Phi_s) \text{ a service defined in } \mathbf{E}$$

Fig. 1. The rules of calculus $\mathcal{SC}$

Essentially, a service definition corresponds to an effective Web service. The service specification provides the formal description of the behavior of the service in terms of pre- and post- conditions. The function Φ_s represents a formal description of service implementation.

Composition of services. The problem of services composition amounts to build a new service from a family of implemented services. We formalize this problem in the context of an *environment*, that is a structure $\mathbf{E} = \langle \mathcal{L}, \mathbf{T}, \eta, \mathcal{S}_1, \dots, \mathcal{S}_n \rangle$ where $\mathbf{T}$ is a theory over the language $\mathcal{L}$, $\eta \in \textsc{it}(\mathbf{T})$, for every $i \in \{1, \dots, n\}$, $\mathcal{S}_i = (s_i, \Phi_i)$ is a service definition in $\mathcal{L}$. $\mathcal{M}$ is a model for $\mathbf{E}$ iff $\mathcal{M} \rhd \langle \eta \rangle\, \mathbf{T}$ and for every $i \in \{1, \dots, n\}$, Φ_i uniformly solves s_i in $\mathcal{M}$. A service specification s' is *solvable* in $\mathbf{E}$ if there exists an implementation Φ' of s' such that, for every model $\mathcal{M}$ of $\mathcal{L}$, if $\mathcal{M}$ is a model for $\mathbf{E}$ then Φ' uniformly solves s' in $\mathcal{M}$.

Now, the main point of service composition is to effectively build the implementation of the service specification starting from the environment. There are two ways to approach the problem: the first one consists in the definition of a composition language which allows the user to build up a new service starting from the environment. The second is given by providing a method to automatically build up the new service implementation. The formalization of the composition problem in the framework of a constructive logic allows to use the proof-theoretical properties of the logical system to support the composition problem. In this paper we concentrate on the definition of a composition language, as for the problem of automatic service composition, it can be seen as a reformulation of the *program-synthesis* problem, which can be solved in $\mathcal{BCDL}$ [4].

The composition calculus $\mathcal{SC}$. The composition calculus we describe in this section is inspired by PAP [6], a calculus which support program synthesis from proofs of a constructive logical system. Our calculus allows to manually compose services guaranteeing the correctness of the composed service.

A *composition* over an environment $\mathbf{E} = \langle \mathcal{L}, \mathbf{T}, \eta, \mathcal{S}_1, \dots, \mathcal{S}_n \rangle$ is defined as:

$$\frac{\mathsf{s}(x) :: P \Rightarrow Q}{\begin{array}{c} \Pi_1 : \mathsf{s_1}(x) :: P_1 \Rightarrow Q_1 \\ \dots \\ \Pi_n : \mathsf{s_n}(x) :: P_n \Rightarrow Q_n \end{array}}\, r$$

- $\mathsf{s}(x) :: P \Rightarrow Q$ is a service specification over $\mathbf{E}$;
- r is one of the rules of the composition calculus $\mathcal{SC}$;
- $\Pi_i : \mathsf{s_i}(x) :: P_i \Rightarrow Q_i$ is a service composition over $\mathbf{E}$ that meets the applicability conditions of r.

The rules of the composition calculus $\mathcal{SC}$ and their computational interpretation (CI) are given in Figure 1. In the rules, the service specification $\mathsf{s}(x) :: P \Rightarrow Q$ is called the *main sequent* of the rule and represents the specification of the service to be composed. The service specifications $\mathsf{s_i}(x) :: P_i \Rightarrow Q_i$ are called *subsequents* of the rule and represent the services involved in the composition. The sequents must satisfy the *applicability conditions* (AC) of the rule, which describe the role the subsequents play in the composition: in the formulation of applicability conditions, $\vdash$ represent derivability in the calculus $\mathcal{ND}$ of [4].

The computational interpretation (CI) of the rules allow to associate with the service composition Π the service implementation Φ_s which is inductively defined on the last rule r applied in Π. In the (CI) rules, given the applicability condition (a) $\Gamma \vdash x : A$ of the rule r, we denote with Φ_a the operator corresponding to the proof π of $\Gamma \vdash x : A$ as defined in [4].

To conclude this section we state the result asserting the soundness of the rules with respect to uniform solvability.

Theorem 1. *Let $\mathbf{E} = \langle \mathcal{L}, \mathbf{T}, \eta, \mathcal{S}_1, \dots, \mathcal{S}_n \rangle$ be an environment and let $\mathsf{s}(x) :: P \Rightarrow Q$ be the main sequent of a composition Π over $\mathbf{E}$. For every model $\mathcal{M}$ for $\mathcal{L}$, if $\mathcal{M}$ is a model for $\mathbf{E}$ then Φ_s extracted from Π uniformly solves s in $\mathcal{M}$.* □

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
2. Bozzato, L., Ferrari, M.: A Note on Semantic Web Services Specification and Composition in Constructive Description Logics. CoRR, abs/1007.2364 [cs.AI] (2010)
3. Bozzato, L., Ferrari, M., Fiorentini, C., Fiorino, G.: A decidable constructive description logic. In: JELIA 2010. LNCS. Springer, Heidelberg (to appear, 2010)
4. Ferrari, M., Fiorentini, C., Fiorino, G.: BCDL: Basic Constructive Description Logic. Journal of Automated Reasoning 44(4), 371–399 (2010)
5. Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing Semantics to Web Services with OWL-S. World Wide Web 10(3), 243–277 (2007)
6. Miglioli, P., Moscato, U., Ornaghi, M.: Program specification and synthesis in constructive formal systems. In: LOPSTR 1991, pp. 13–26. Springer, Heidelberg (1991); Workshops in Computing

A Step toward Tight Integration of Fuzzy Ontological Reasoning with Forward Rules

Stefano Bragaglia, Federico Chesani, Paola Mello, and Davide Sottara

DEIS – University of Bologna,
Viale Risorgimento, 2
40136 - Bologna, Italy
name.surname@unibo.it

Abstract. Integrating distinct reasoning styles such as the ones exploited by description logics, rule-based systems and fuzzy logic is still an open challenge because of the differences among them. Three complementary approaches suggest possible models of integration: loose integration, tight integration and embedded integration. Loose integration couples existing tools into a hybrid system, handling their mutual interactions and keeping their knowledge aligned. Tight integration, instead, is based on a unique theory and framework supporting both reasoning styles. Embedded integration is a mixed approach aimed to the simplicity of the former and the efficiency of the latter. In this paper we present our experiences on the implementation of a basic loosely-coupled system and a more advanced embedded solution.

Keywords: Rule-based Systems, Description Logics, Tableau Reasoning, Fuzzy Systems.

1 Introduction and Motivations

Recently, the possibility of combining the descriptive capacity of Description Logics with the operational semantics of rule-based systems within the same application domain has been investigated with growing interest. The Description Logics and Semantic Web's tools provide formal languages to represent knowledge and algorithms to detect inconsistencies, to classify concepts and to recognize individuals with respect to the given knowledge. Similarly rule-based systems, and production rule systems in particular, have been widely used to express the application logic on a domain with rules that produce the expected outcomes by triggering on external stimuli.

Although those technologies individually are relatively mature, the need for a comprehensive and unifying approach is emerging. In fact some domains that would greatly benefit from such an approach, have been already identified in literature [1,6,8]. In the context of *eTourism*, for example, tour operators are used to write rules to handle offers. They also tend to define new offers as specializations (subclasses) of the existing ones. In this way, existing rules cannot trigger on such subclasses unless a suitable rules refactoring takes place. An hybrid system, on the other hand, could exploit ontological reasoning to make the rules trigger without further changes.

Moreover, the need to represent real-life domains with an appropriate level of expressiveness is also emerging. At the moment, in fact, the Semantic Web proposals can only support crisp concepts and formulas while many domains require to express concepts in terms of shaded degrees of truth, according to the Fuzzy Logic's semantics. Consider for example that learning vacations should be only recommended to the parents of young customers. Determining how young is a customer is not a trivial task because the degree of truth of this property gradually changes with her age. Therefore, a system which natively handles such kind of fuzzy concepts is preferable.

At the moment, the technological integration of production rule and Description Logics systems has been attempted within frameworks such as Jena[1], Algernon[2] and Sweet-Rules[3]. The integration of fuzzy logics within rule-based systems has been investigated with FuzzyClips[4] and FuzzyJess[5]. Finally, extensions of Description Logics with fuzzy constructs and semantics has been applied in DeLorean [2] and in FuzzyDL [3]. To the best of our knowledge, however, no tool supporting ontological reasoning, rule-based reasoning, and fuzzy reasoning at the same time is currently available.

2 Integration Issues

In general, the integration of different reasoning styles is usually a rather difficult problem. When combining Description Logics-based and rule-based reasoning, for example, the primary problem concerns to their different contextual hypothesis [9].

Rule-systems, in fact, typically embrace to the *Closed World Assumption* (CWA) which basically allows to consider false anything not explicitly asserted as true. This kind of inference is non-monotonic, meaning that each new conclusion may invalidate part of the knowledge base if the available information is not relatively complete. Description Logics usually adhere to *Open World Assumption* (OWA) which allows to consider true (or false) only the conclusions that may be safely derived from current knowledge. This approach is more cautious: it never contradicts previous knowledge but may lead to non-determinism. Problems arise when the integrated system has to combine nondeterministic and deterministic results.

In the context of Semantic web, the integration with fuzzy reasoning has only to cope with the family of truth functional many-valued logics rather than with the broader context of definitions expressed using vague linguistic terms. Anyway, the integration of multi-valued logics in boolean, forward-chaining rule-based

[1] http://jena.sourceforge.net/
[2] http://algernon-j.sourceforge.net/
[3] http://sweetrules.semwebcentral.org
[4] http://www.nrc-cnrc.gc.ca/eng/projects/iit/fuzzy-reasoning.html
[5] http://www.csie.ntu.edu.tw/~sylee/courses/FuzzyJ/FuzzyJess.htm

systems usually complicates the conflict resolution strategies. The propagation of computed truth values, in fact, may impose precedence relations between rules which should be independent instead. Similarly the integration of fuzzy definitions with Description Logics may increase the complexity of the satisfiability check algorithms, making some domains difficult to be treated.

3 Practical Implementations

From the implementation viewpoint, integration may be achieved by two complementary approaches: *loose integration* vs. *tight integration*. Loosely integrated systems bring together an appropriate mix of existing tools for the desired reasoning styles within a common interface. Despite each single tool may work to its full potential, consistency issues among them may reduce the overall expressiveness of the whole system. Tightly integrated system, instead, implement a new coherent theory which is complex enough to provide a language and algorithms to support the planned reasoning tasks. In this case, less design constraints should allow more expressiveness but often the overall complexity is likely to reduce it. There is also a third approach, namely *embedded integration*, which is tries to implant one or more reasoning modules within another [7]. Depending on their embedding degree, they can be considered as a mediation between the other two approaches.

As a first attempt, we decided to implement a hybrid prototype based on a loose integration approach [4]. This prototype featured Drools[6] (a popular open-source production rule system with capabilities similar to RIF's Production Rule Dialect), Pellet[7] and FuzzyDL[8]. It also exploited Jena and JenaBean[9] APIs to dispatch reasoning request among modules, granting the coherence of their models. Despite its simplicity, the solution required the distinct kinds of knowledge to be stated separately, asking the user to be consistent each time. Moreover, dealing with such separate models makes the system inefficient in terms of both memory usage and entailment.

Those considerations led us to consider a tighter approach: due the operational nature of its definition, we decided to implement a tableau algorithm (typically used to perform reasoning in Description Logics) by means of Drools rules [5]. The introduction of fuzzy logic expressiveness required the inclusion of an external MILP-solver to compute fuzziness and this allowed us to generate a tableau whose expansion is deterministic. In general, in fact, tableaux may be non-deterministic: a behaviour which is difficult to emulate in a production rule system. Thanks to that solution, we were able to use a single model for the whole system. Such a model contains facts annotated with fuzzy truth intervals which are used by the tableau to answer queries with results that are fuzzy intervals themselves. Notably, we can cast back the intervals to "traditional" fuzzy values (both in an OWA and a CWA sense) or even to boolean values using dedicated operators.

[6] http://www.jboss.org/drools
[7] http://clarkparsia.com/pellet
[8] http://gaia.isti.cnr.it/~straccia/software/fuzzyDL/fuzzyDL.html
[9] http://code.google.com/p/jenabean/

Architectural details were not provided here for lack of space. More details on prototypes can be found at http://ai.unibo.it/projects.

4 Conclusions

We believe that the Semantic Web will change many aspects of every-day life in the years to come, as the Web already did. We also think that semantic tools which can handle the application logic of real-life domains with fuzzy expressiveness, like the proposed ones, will help make the change.

At the moment, our embedded approach can only treat the fuzzy $\mathcal{ALC}$ fragment of Description Logics, but we plan to possibly extend it to OWL-DL. In effect, more advanced constructors could complicate the tableau, eventually leading to much harder problems than MILP. These problems, however, can be easily solved with a more powerful solver. We are also predisposing a common dataset, suitable for both versions, on which to conduct experiments to evaluate their performances. Nevertheless we feel that the unique features of our proposal could contribute to the Semantic Web revolution.

Acknowledgement. This work has been partially supported by the Italian MIUR PRIN 2007 project No. 20077WWCR8.

References

1. Analyti, A., Antoniou, G., Damásio, C.: A principled framework for modular web rule bases and its semantics. In: Proc. of 11th Intl. Conference on Principles of Knowledge Representation and Reasoning, KR, pp. 390–400 (2008)
2. Bobillo, F., Delgado, M., Gómez-Romero, J.: DeLorean: A reasoner for fuzzy OWL 1.1. In: Proc. of the 4th Intl. Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2008), CEUR Workshop Proc., Citeseer, vol. 423 (2008)
3. Bobillo, F., Straccia, U.: fuzzyDL: An expressive fuzzy description logic reasoner. In: IEEE Intl. Conference on Fuzzy Systems, FUZZ-IEEE 2008 (IEEE World Congress on Computational Intelligence), pp. 923–930 (2008)
4. Bragaglia, S., Chesani, F., Ciampolini, A., Mello, P., Montali, M., Sottara, D.: An Hybrid Architecture Integrating Forward Rules with Fuzzy Ontological Reasoning. In: Graña Romay, M., Corchado, E., Garcia Sebastian, M.T. (eds.) HAIS 2010. LNCS, vol. 6076, pp. 438–445. Springer, Heidelberg (2010)
5. Bragaglia, S., Chesani, F., Mello, P., Sottara, D.: A Rule-Based Implementation of Fuzzy Tableau Reasoning. Submitted to RuleML (2010)
6. Damásio, C., Analyti, A., Antoniou, G., Wagner, G.: Supporting open and closed world reasoning on the web. In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) PPSWR 2006. LNCS, vol. 4187, pp. 149–163. Springer, Heidelberg (2006)
7. De Bruijn, J., Bonnard, P., Citeau, H., Dehors, S., Heymans, S., Korf, R., Pührer, J., Eiter, T.: State-of-the-art survey of issues. Tech. rep., ONTORULE (2009)
8. Polleres, A., Feier, C., Harth, A.: Rules with contextually scoped negation. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 332–347. Springer, Heidelberg (2006)
9. Reiter, R.: On closed world data bases. Morgan Kaufmann Publishers Inc., San Francisco (1987)

Rule-Based Spam E-mail Annotation*

Giacomo Fiumara, Massimo Marchi, Rosamaria Pagano, and Alessandro Provetti

Dept. of Physics, Informatics section. University of Messina,
Viale F. Stagno d'Alcontres, 31. I-98166 Messina, Italy
{gfiumara,mmarchi,rpagano,ale}@unime.it

Abstract. A new system for spam e-mail annotation by end-users is presented. It is based on the recursive application of hand-written annotation rules by means of an inferential engine based on Logic Programming. Annotation rules allow the user to express nuanced considerations that depend on deobfuscation, word (non-)occurrence and structure of the message in a straightforward, human-readable syntax. We show that a sample collection of annotation rules are effective on a relevant corpus that we have assembled by collecting e-mails that have escaped detection by the industry-standard SpamAssassin filter. The system presented here is intended as a personal tool enforcing personalized annotation rules that would not be suitable for the general e-mail traffic.

1 Introduction

This article describes a rule-based *personal* e-mail annotator that works as a spam sentinel. The annotator applies given set of annotation rules (called *annotation policy)* by means of a Prolog inferential engine. Our annotator is called RuBaST , for *Rule-Based Spam Terminator* and is intended as a filtering layer between a typical, site-wide spam filter, e.g., SpamAssassin [1] and the user's mailbox.

The RuBaST e-mail annotator works as follows. First, it reads the e-mail file and applies ad-hoc regular expressions that de-obfuscate suspicious words, e.g., *Ciaa1lis* is rendered as *Cialis.* Next, the whole e-mail is converted into a set of Prolog facts. Then Prolog rules representing the user's annotation policy are combined with those facts. Annotation rules describe, essentially, a scoring system. The annotation rules in Prolog syntax are applied by means of the The SWI-Prolog inferential engine is invoked to attempt, recursively, to apply each rule to each e-mail representation. The overall score is obtained as the sum of all scores given by the rules; if it exceeds a user-defined threshold the e-mail is flagged as spam in the subject.

RuBaST annotation rules reflect a user's personal, if idiosyncratic, view of the relevance of a mail. For instance, these authors are receiving, even after SpamAssassin filtering, several messages posing as coming from the Web site of the Italian Postal system. As we have no account there, it remained easy to issue a RuBaST rule giving high spam likeliness to such messages. Such rule would not be appropriate for a site-wide spam filter, for obvious reasons. An example that would prove hard to encode (for any

* A companion Web site to this article, with software, results and the corpus described herewith is at http://informatica.unime.it/rubast/

system) is that of e-mails carrying call for papers that are sent outside the usual distribution lists and relative to conferences that are unknown, out of scope or in a foreign language.

2 The Implementation

The implementation of RuBaST consists mainly of a Java application with a Prolog-based automated reasoning core. The interface between the two modules is performed by the JPL package provided by SWI-Prolog. Figure 1 summarizes the architecture of our implementation. Such architecture addresses two main concerns that have emerged during our project. First, the current solutions for integrating inferential engines into a Java program (e.g., tuProlog [2]) were considered not viable for this project or harder to use vis-à-vis SWI Prolog. Second, we would like to retain the ability to experiment with different inferential engines in the future.

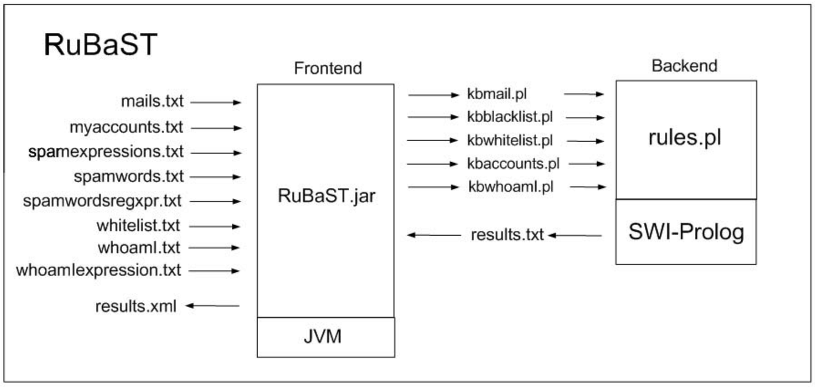


Fig. 1. The overall architecture

JPL [3] is a library that uses the SWI-Prolog[1] foreign interface and the Java JNI interface[2] to interleave Prolog and Java executions. The rôle of the Java module, then, is to prepare the message information for the Prolog engine. We call this preparation phase *normalization*. The main task of normalization is the so-called *de-obfuscation* of the mail body, i.e., spotting (and replacing) terms that have been disguised by the spammer in order to hide them from the anti-spam filters.

2.1 The Prolog Module

The Prolog module represents the core of our anti-spam filter. It applies two filters:

[1] `http://www.swi-prolog.org/`
[2] `http://java.sun.com/j2se/1.4.2/docs/guide/jni/`

WORDS_FILTER, which operates on the *Subject* and *Body* of the message, and

RULES_FILTER, which operates on *envelope* informations such as sender and addressee (*From:* and *To:*).

The final score consists of the combination of the scores recommended by each filter.

Rules for analyzing the content. A set of Prolog rules have been defined to find the occurrence of black words in e-mails (presented as simple lists of normalized words) and to give a score based on co-occurrences of blackwords and their distance (defined simply as the number of words that occur in between).

Currently, 11 annotation rules capture more sofisticated forms of reasoning about the content. As a way of illustration, let us see in some detail the rule that reasons about contextual occurrence or co-occurrence of specialized terms normally associated to legitimate messages. Indeed, in several cases, deobfuscation may lead to undesired results, whereas considering the context of the occurrence may lead to a correct classification, as illustrated in the following example.

The address used for collecting our corpus regularly receives legitimate e-mails that mention the so-called *Laurea specialistica,* an Italian degree roughly equivalent to a MSc/MEng. Unfortunately, the words *specialistica* contains the word *cialis* properly, hence deobfuscation renders it as cialis, which in turns triggers blackword-based annotation rules to give it a very high score. An exception has been issued; the syntax is as follows:

```
spam_word('cialis', 85).
contextual_exception('cialis',['laurea', ...]).
```

3 Experimental Validation

We have tested the first implementation of RuBaST against a corpus of e-mails that have been collected, to this purpose, by just one e-mail address over a several months period. Upon reception, the user gave an *exact* classification w.r.t. to the fine-grained categorizazion of spam (and *ham,* i.e., legitimate e-mails) proposed by Cormack et al. [4] to facilitate spam filter evalutaion. In separate sessions, the recipient also wrote the annotation rules that were passed to RuBaST during the experimental validation. We submit that:

- our corpus, as apposed to existing public corpora (e.g. TREC[3]) has been collected at a single e-mail address, so it arguably more adequate for evaluating a personalized spam filter, and
- by having escaped Spamassassin filtering with low threshold at the site-level, messages in our corpus are deemed challenging. They arguably fall in the 98th percentile of spam detection hardness, according to the recent evaluation by [4].

To validate the effectiveness of RuBaST in weeding out spam messages, we have tested it in terms of Information Retrieval (IR) [5], i.e., as a challenge for RuBaST to select

[3] The TREC Public Spam corpus is at `http://trec.nist.gov/`

all and only the messages in the corpus that were deemed spam. Then, we have compared each annotation to the *exact* classification given by the recipient and computed the overall score according to the Cormack et al. metrics [4], which are now standard for spam filtering evaluation.

4 Discussion and Related Work

RuBaST is a novel application of rule-based reasoning. Our experimental assessment shows that it can be effective in weeding out spam e-mails on a user-by-user basis. With our work we believe to have shown that -rather than replacing the main spam ML-based filters— Prolog-based spam annotation can be effective when annotation policies need to be explicit, transferable and (de)composable. With the exception of SpamPIG[4], an unpublished graduate project, we are not aware of similar efforts in the Logic Programming and AI domain. In our work on setting up the personalized annotator, we have deliberately refrained from adopting a uniform methodology in the creation of rules. The rationale was to leave room for *unexpected* combinations of conditions, mixing e.g., word occurrence with the suffix of the e-mail address, that would not be easily discovered had we adopted some methodical approach to writing rules.

The assessment was done on a coherent (same recipient for all messages), medium-sized corpus (as opposed to large but very diversified corpora, e.g., TREC [6] and SpamAssassin.). Yet, our corpus is relevant precisely because it has been collected at a sole e-mail address, which is the intended use case: RuBaST enforces personalised annotation policies against the particular spam attack one recipient is under. We will release our corpus as a service to the community in the same context of this article.

To summarize the information retrieval validation: RuBaST can halve the amount of spam received, while only few (around 1%) regular e-mails might be misclassified. From the point of view of the core annotation operations, a future improvement concerns the introduction of a *word whitelist* to reduce the possibility of false positives; e.g., the keyword *CEAS* (acronym of the international conference on fighting spam) might counterbalance the occurrences of *spam* in the same message.

References

1. Sergeant, M.: Internet-level spam detection and spamassassin 2.50. In: Spam Conference (2003)
2. Denti, E., Omicini, A., Ricci, A.: Multi-paradigm java-prolog integration in tuprolog. Sci. Comput. Program. 57(2), 217–250 (2005)
3. Wielemaker, J., Anjewierden, A.: An architecture for making object-oriented systems available from prolog. In: Proc. of the 12th Int'l Workshop on Logic Programming Environments, WLPE 2002 (2002)
4. Cormack, G.V., Lynam, T.R.: Online supervised spam filter evaluation. ACM Trans. Inf. Syst. 25(3) (2007)
5. van Rijsbergen, C.J.: Information Retrieval, 2nd edn. Butterworths, London (1979)
6. Cormack, G.V., Lynam, T.R.: Spam corpus creation for trec. In: Proc. of the Second Conference on Email and Anti-Spam, CEAS 2005 (2005)

[4] `http://spampig.sourceforge.net/`

A RESTful SWRL Rule Editor

Carsten Keßler

Institute for Geoinformatics, University of Münster, Germany
carsten.kessler@uni-muenster.de

Abstract. The sparse application of the Semantic Web Rule Language is partly caused by a lack of intuitive rule editors. This applies both from a human user's, as well as from a software interoperability perspective, as creating and modifying rules is currently hard in distributed Web applications. We introduce a Representational State Transfer-based approach that enables online rule editing to overcome these problems.

1 Introduction and Motivation

Rule-based reasoning still plays a minor role on the Semantic Web, despite the gain in expressivity[1] offered by the Semantic Web Rule Language (SWRL) [2]. While rules provide powerful solutions for problems that cannot be solved with standard Description Logic-based reasoning [3], the verbose syntax and a lack of intuitive user interfaces for rule editing hamper their application. Moreover, proprietary application programming interfaces complicate the integration into standards-based systems. We propose a wrapper for rule engines based on the Representational State Transfer (REST) approach [4] to overcome these problems. REST offers standardized, straight-forward access to resources without unnecessary overhead. Masking the complexity of rule editing as a RESTful service makes rule-based reasoning available for a wide range of services. At the same time, rule functionality is made more accessible for human users, as developers can reduce the editing options to the subset of SWRL's expressivity required for a specific application. Changes to single atoms can be completed without confronting users with the (potentially lengthy) complete rule, and adapted results can be directly returned in the server response. Besides the mapping approach from REST URIs to rules, we discuss the architecture of this RESTful wrapper. Moreover, we demonstrate its application in a mobile tool for personalized information retrieval that uses rules to represent user preferences.

2 Mapping REST to SWRL

Rule editing comprises creating, updating and deleting, in addition to simple access to the rules. These activities apply both for complete rules as well as for single atoms in the rules' bodies or heads. REST makes use of the `HTTP` request methods `GET`, `POST`, `DELETE` and `PUT` to represent these activities. Resources are represented by descriptive URIs and can be queried and modified using these methods. Table 1 shows the required URIs for a complete rule editor.

[1] For OWL 2, the additional expressivity is partly covered in the RL profile [1].

Table 1. URIs for rule editing. The last five URIs also apply for `head` atoms.

Resource URI	HTTP Method	Description
`/rules`	`GET`	Lists all rule IDs
`/rules`	`POST`	Creates empty rule, returns ID
`/rules/{id}`	`GET`	Returns the rule with this ID
`/rules/{id}`	`DELETE`	Deletes the specified rule
`/rules/{id}/body`	`GET`	Lists IDs of all atoms in this rule's body
`/rules/{id}/body`	`POST`	Adds new atom to body, returns ID
`/rules/{id}/body/{id}`	`GET`	Returns a specific atom
`/rules/{id}/body/{id}`	`PUT`	Overwrites a specific atom
`/rules/{id}/body/{id}`	`DELETE`	Deletes a specific atom

Figure 1 gives an overview of the wrapper architecture. It creates a transparent access point for the KnowledgeBaseManager, providing access to the ontology repositories building the application knowledge base, and the RuleEngine. In case of the prototype implementation, Protégé Core has been used for knowledge base management, and Jess for rule execution[2]. The rule engine maintains the connection to the ontology and exposes its concepts and relations to the wrapper, which maps them to their respective URIs. Clients can modify rules via these URIs (see Table 1). When existing rules are modified, the (application-dependent) server response contains the knowledge inferred after rule execution. The rule engine also checks rules for validity before execution; if errors occur due to faulty client input, these must be passed on to the wrapper and forwarded to the client with the respective `HTTP` status codes.

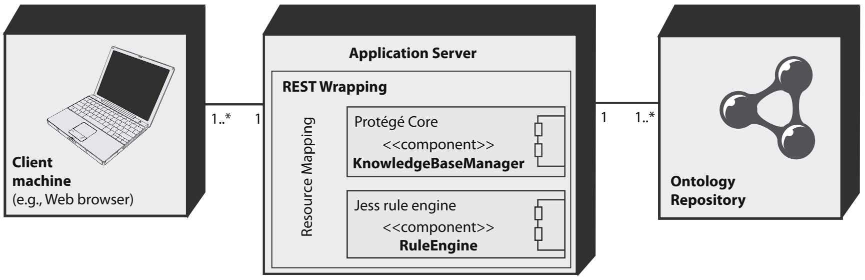


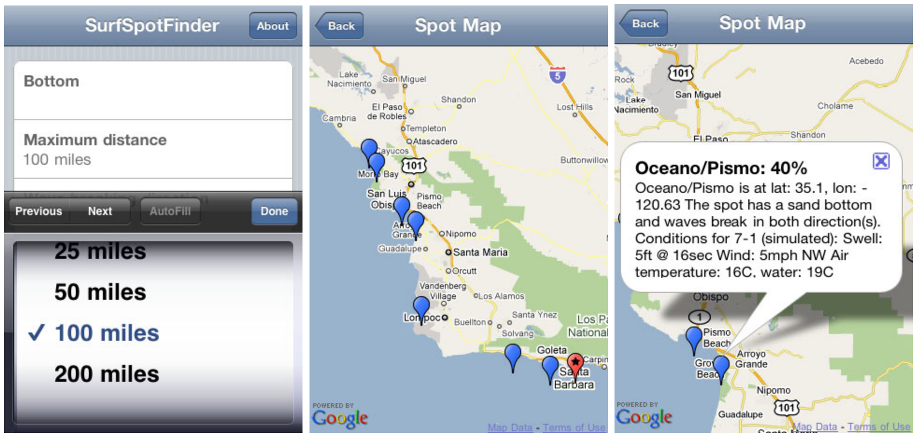
Fig. 1. Deployment diagram for the RESTful wrapper; adapted from [5]

3 Prototype: The Surf Spot Finder

The RESTful approach for SWRL rule editing described in Section 2 has been tested in the Surf Spot Finder [6], a mobile Web application for personalized recommendations for surf spots at California's central coast. The Web interface

[2] See `http://protege.stanford.edu` and `http://jessrules.com`

(see Figure 2) allows users to set their preferences for different aspects, each of which is mapped to one atom in the rule representing the user's profile. Upon rule execution, all surf spots instances in the knowledge base that match the user preferences are reclassified as appropriate for this user. In order to maintain the link of a user to her specific rule, the user's ID is stored in a cookie on the client side. On the server side, a new resource is created for every user, based on her ID (see Table 2). This maintains the stateless nature of REST, as no session data are stored, yet it allows users to preserve their setting between uses.



Oceano/Pismo: 40%
Oceano/Pismo is at lat: 35.1, lon: -120.63 The spot has a sand bottom and waves break in both direction(s). Conditions for 7-1 (simulated): Swell: 5ft @ 16sec Wind: 5mph NW Air temperature: 16C, water: 19C

Fig. 2. Screen shots of the prototype user interface [5]

Table 2. Resources offered by the Surf Spot Finder prototype. The three operations for `bottom` are also available for all other aspects of the user profiles.

Resource URI	Method	Description
`/users`	GET	Returns a list of all users' rules
`/users`	POST	Creates new user, returns ID
`/users/{id}`	GET	Returns a user's SWRL rule
`/users/{id}`	DELETE	Deletes the specified user profile
`/users/{id}/spots`	GET	Returns matching spots
`/users/{id}/bottom`	GET	Returns the user's preferred bottom type
`/users/{id}/bottom`	PUT	Updates user's preferred bottom type, returns spots
`/users/{id}/bottom`	DELETE	Deletes user's preferred bottom type, returns spots

Imagine the following rule represents user 42's preferences: `SurfSpot(?spot)` $\wedge$ `hasBottom(?spot, 'rock')` $\rightarrow$ `Match(?spot)`. A PUT request to `http://somedomain.com/users/42/bottom` with `sand` as contents overwrites the corresponding atom, changing the rule to `SurfSpot(?spot)` $\wedge$ `hasBottom(?spot, 'sand')` $\rightarrow$ `Match(?spot)`. Execution of the rule reclassifies all matching SurfSpot instances as a Match, which are then returned to the client by the wrapper. Communication with the server component is asynchronous, allowing the application to update the map without reloading the whole application. When the user changes one of the parameters in her profile, this change is sent to the server and the updated set of matching spots is returned and shown on the map.

4 Conclusions and Future Work

We have proposed a RESTful Web service that enables online editing of SWRL rules. The general approach has been introduced and demonstrated in the Surf Spot Finder application that uses SWRL-based user profiles. The next steps in this research are the implementation of a *complete* mapping for rule editing as outlined in Section 2. Two aspects of this implementation are especially challenging. First, the automatic mapping of any built-ins, especially custom built-ins [7]. Second, an integrity checking for rule modifications that wraps potential error messages with the appropriate `HTTP` status codes.

Acknowledgments

This work has been funded by the German Research Foundation's *SimCat* project (DFG Ra1062/2-1 and Ja1709/2-2), see `http://sim-dl.sourceforge.net`.

References

1. W3C: OWL 2 Web Ontology Language – Profiles. W3C Recommendation (October 27, 2009) `http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/`
2. W3C: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission (May 21, 2004), `http://w3.org/Submission/2004/SUBM-SWRL-20040521/`
3. Horrocks, I.: OWL rules, OK? In: W3C Workshop on Rule Languages for Interoperability (2005)
4. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, USA (2000)
5. Keßler, C.: Context-aware Semantics-based Information Retrieval. PhD thesis, Institute for Geoinformatics, University of Münster, Germany (May 2010)
6. Keßler, C., Raubal, M., Wosniok, C.: Semantic Rules for Context-Aware Geographical Information Retrieval. In: Barnaghi, P., Moessner, K., Presser, M., Meissner, S. (eds.) EuroSSC 2009. LNCS, vol. 5741, pp. 77–92. Springer, Heidelberg (2009)
7. O'Connor, M., Das, A.: A Mechanism to Define and Execute SWRL Built-ins in Protégé-OWL. In: 9th Int. Protégé Conference, Stanford, California, July 23-26 (2006)

On the Termination of the Chase Algorithm

Michael Meier

University of Freiburg, Georges-Köhler-Allee 051, 79110 Freiburg, Germany
meierm@informatik.uni-freiburg.de

Abstract. In my PhD thesis I study the termination problem of the chase algorithm, a central tool in various database problems such as the constraint implication problem, conjunctive query optimization, rewriting queries using views, data exchange, and data integration.

1 Introduction

The chase procedure is a fundamental algorithm that has been successfully applied in a variety of database applications [5,12,3,11,6,8,9,1]. The basic idea of the chase is, given a database instance and a set of constraints as input, to fix constraint violations in the database instance. It is well-known that for an arbitrary set of constraints the chase does not necessarily terminate. In general, it is undecidable whether the chase terminates, given a set of TGDs as input, on the empty database [7]. Addressing this issue, we review the limitations of existing sufficient termination conditions for the chase and develop new techniques that allow us to establish weaker sufficient conditions. For the first time in the literature, we develop methods that allow us to ensure the termination of at least one chase sequence and not necessarily of all. We then study the interrelations of our termination conditions with previous conditions and the complexity of checking them. As another contribution, we study the problem of data-dependent chase termination and present sufficient termination conditions with respect to fixed instances. They might guarantee termination when our data-independent techniques cannot. As application of our techniques beyond those already mentioned, we transfer our results into the field of semantic query optimization in the presence of types.

2 Chase Termination

We divide the problem of chase termination into two branches. The first one studies data-independent methods. As the term data-independent suggests, these chase termination conditions work for every database instance. In contrast the second branch studies methods that depend on a given database instance, which is why they are called data-dependent. Both branches are divided again into two subbranches, namely sequence-independent and sequence-dependent methods. Sequence-independent chase termination conditions can guarantee chase termination no matter what chase sequence has been taken during the execution of the chase, in difference to that sequence-dependent conditions give termination guarantees only for (at least) one chase sequence. So far, the only branch that has been considered in the literature are the data- and sequence-independent methods.

We want to point out that my thesis is the first study of sequence-dependent methods and data-dependent termination conditions at all. In the following we summarize the key concepts and ideas of our analysis and survey the main results.

2.1 Sequence-Dependent Termination: $\text{CT}_{\forall\exists}$

$\text{CT}_{\forall\exists}$ is the class of TGDs and EGDs that ensures the existence of at least one terminating chase sequence for every database instance. The literature on the chase lacks a systematic study of $\text{CT}_{\forall\exists}$ and concentrates solely on sequence-independent termination. We fill this gap by identifying decidable fragments of it in [16,14]. To the best of our knowledge, these papers are the first study of sufficient termination conditions for the chase that do not ensure the termination of all chase sequences but of at least one.

But how can we safely apply the chase and be sure that the chase sequence we follow terminates? How can such a terminating sequence be found if we know that it exists? A general solution seems to be quite simple. We apply the chase in a breadth-first manner.[1] Using this technique, we can guarantee termination. However, this method is quite inefficient because it needs a lot of runtime to follow all chase sequences and it also uses a lot of memory.

The results of our study on sequence-dependent chase termination have an important additional property. We cannot only ensure that there is a terminating chase sequence, but we can statically determine it, while checking our termination conditions. This has an important implication. We do not have to apply the chase in the breadth-first, but in the usual depth-first manner, thus saving much time and space.

We show that stratification introduced in [7] does not generally ensure termination of every chase sequence, as stated by the authors of [7], but of at least one chase sequence. Besides, we show that such a sequence can be statically determined independently of the input instance. Furthermore, we propose a possible correction of the stratification condition which ensures sequence-independent chase termination, as intended by the authors of [7], using the oblivious chase [4,13]. This correction forms the basis for nearly all further results on sequence-independent chase termination.

2.2 Summary of Results on $\text{CT}_{\forall\forall}$

$\text{CT}_{\forall\forall}$ is the class of TGDs and EGDs such that the chase terminates for every possible input database instance and every chase sequence. The main idea of almost all of our termination conditions is as follows. The reason for infinite chase sequences is an infinite cascading of labeled nulls. Therefore our idea is, given a constraint violation $I \not\models \alpha(\overline{a})$ during the application of the chase algorithm, we try to estimate what values in $\overline{a}$ are null values that were newly introduced by the chase algorithm and have not been in the input instance. We provide different kinds of approximation which are used by our termination conditions.

Figure 1 surveys our main results on $\text{CT}_{\forall\forall}$ from [14] and relates them to the previous termination condition weak acyclicity and to c-stratification, the corrected version

[1] Applying the chase breadth-first means that for all $n = 1, 2, 3, ...$ we generate all possible chase sequences of length bounded by n and we abort this process when we have found a terminating sequence.

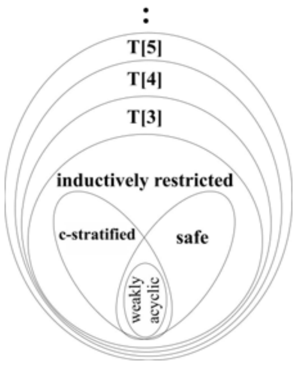


Fig. 1. Data- and sequence-independent chase termination conditions

of stratification. Please note that our paper [14] contains a corrected version of our results from [15] that we published due to an error that we made in [15]. All inclusion relationships in Figure 1 are strict. As it can be seen, safety generalizes weak acyclicity and is further generalized by inductive restriction. On top of inductively restricted constraints we ultimately define a hierarchy of sufficient termination conditions called T-hierarchy. To give an intuition for a fixed class in this hierarchy, say T$[k]$, we study the flow and creation of fresh null values in detail for chains of up to k constraints that might cause to fire each other in sequence.

We also want to mention that there other authors working in the field of chase termination especially CT$_{\forall\forall}$, e.g. [13] develops the notion of super-weak acyclicity that guarantees membership in CT$_{\forall\forall}$ and termination of the oblivious skolem chase. In [10] the authors rewrite the given constraint set such that termination for the original set can be guaranteed if the rewritten constraint set satisfies a termination condition. Furthermore, in [4,2] the authors consider the problem of deciding query containment under constraints for cases when the chase does not necessarily terminate.

2.3 Data-Dependent Termination

So far, we have discussed conditions that guarantee chase termination for every database instance. In [15,14] we study the problem of data-dependent termination, i.e. given the constraint set Σ and a fixed instance I, does the chase with Σ terminate on I?

Reasonable applications should not risk non-termination, so for some constraint sets chase termination is in question for all queries, although there might be queries for which the chase terminates.[2] Tackling this problem, we propose to investigate data-dependent chase termination, i.e. to study sufficient termination guarantees for a fixed instance when no general termination guarantees apply.

[2] Note that query optimization can be done with a bounded portion of a chase result but in general we do not find minimal rewritings of the input query in the style of [1]. Therefore, it is desirable to guarantee chase termination.

Given a fixed database instance, our first idea is to exclude constraints from our constraint set that may never fire during chase application. We can test our data-independent methods on the remaining constraint set and if this test succeeds we can ensure chase termination for our fixed database instance.

If the previous data-dependent termination conditions do not apply, we propose to monitor the chase run and abort if tuples that may potentially lead to non-termination are created. We introduce a data structure called monitor graph that allows us to track the chase run. We abort chase application if two many potentially bad tuples are created.

3 Semantic Query Optimization in the Presence of Types

Both semantic and type-based query optimization rely on the idea that queries often exhibit non-trivial rewritings if the state space of the database is restricted. Despite their close connection, these two problems to date have always been studied separately. We present a unifying, logic-based framework for query optimization in the presence of data dependencies and type information [16]. It builds upon the classical chase algorithm and extends existing query minimization techniques to considerably larger classes of queries and dependencies. As a technical challenge, our setting involves chasing of conjunctive queries in the presence of constraints containing negation and disjunction.

In response to the central role of chase termination in our setting, we develop novel chase termination conditions for constraint sets involving disjunction and negation. Rather than developing these termination conditions from scratch, our approach is to carry over existing termination conditions for standard TGDs and EGDs, i.e. we show how to rewrite our constraint set to TGDs and EGDs such that if the chase terminates for the rewritten constraint set, then it must also terminate for the original one.

References

1. Deutsch, A., Popa, L., Tannen, V.: Query Reformulation with Constraints. SIGMOD Record 35(1), 65–73 (2006)
2. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, Ê.: Extending decidable cases for rules with existential variables. In: IJCAI, pp. 677–682 (2009)
3. Beeri, C., Vardi, M.Y.: A Proof Procedure for Data Dependencies. J. ACM 31(4), 718–741 (1984)
4. Calì, A., Gottlob, G., Kifer, M.: Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In: KR (2008)
5. Maier, D., Mendelzon, A., Sagiv, Y.: Testing Implications of Data Dependencies. In: SIGMOD, pp. 152–152 (1979)
6. Deutsch, A., Ludäscher, B., Nash, A.: Rewriting Queries Using Views with Access Patterns under Integrity Constraints. Theor. Comput. Sci. 371(3), 200–226 (2007)
7. Deutsch, A., Nash, A., Remmel, J.: The Chase Revisited. In: PODS, pp. 149–158 (2008)
8. Fagin, R., Kolaitis, P., Miller, R., Popa, L.: Data Exchange: Semantics and Query Answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
9. Fuxman, A., Kolaitis, P., Miller, R., Tan, W.-C.: Peer Data Exchange. TODS 31(4), 1454–1498 (2006)
10. Greco, S., Spezzano, F.: Chase Termination: A Constraints Rewriting Approach. To appear in PVLDB (2010)

11. Halevy, A.: Answering Queries Using Views: A Survey. VLDB Journal 10(4), 270–294 (2001)
12. Johnson, D., Klug, A.: Testing Containment of Conjunctive Queries Under Functional and Inclusion Dependencies. In: PODS, pp. 164–169 (1982)
13. Marnette, B.: Generalized Schema-Mappings: From Termination to Tractability. In: PODS, pp. 13–22 (2009)
14. Meier, M., Schmidt, M., Lausen, G.: On Chase Termination Beyond Stratification (2009), `http://arxiv.org/abs/0906.4228`
15. Meier, M., Schmidt, M., Lausen, G.: On Chase Termination Beyond Stratification. PVLDB 2(1) (2009)
16. Meier, M., Schmidt, M., Wei, F., Lausen, G.: Semantic Query Optimization in the Presence of Types. In: PODS (2010)

Processing RIF and OWL2RL within DLVHEX*

Marco Marano[1], Philipp Obermeier[2], and Axel Polleres[2]

[1] Universita' della Calabria
mmarano@deis.unical.it
[2] Digital Enterprise Research Institute, National University of Ireland, Galway
{philipp.obermeier,axel.polleres}@deri.org

Abstract. We present an extension of the DLVHEX system to support RIF-Core, a dialect of W3C's Rule Interchange Format (RIF), as well as combinations of RIF-Core and OWL2RL ontologies. DLVHEX is a plugin system on top of DLV, a disjunctive Datalog engine which enables higher-order and external atoms, as well as input rewriting capabilities, which are provided as plugins and enable DLVHEX to bidirectionally exchange data with external knowledge bases and consuming input in different Semantic Web languages. In fact, there already exist plugins for languages such as RDF and SPARQL. Our new plugin facilitates consumption and processing of RIF rule sets, as well as OWL2RL reasoning by a 2-step-reduction to DLVHEX via embedding in RIF-Core. The current version implements the translation from OWL2RL to RIF by a static rule set [12] and supports the RIF built-ins mandatory for this reduction trough external atoms in DLVHEX. For the future we plan to switch to a dynamic approach for RIF embedding of OWL2RL [2] and extend the RIF reasoning capabilities to more features of RIF-BLD. We provide a description of our current system, its current development status as well as an illustrative example, and conclude future plans to complete the Semantic Web library of plugins for DLVHEX.

1 Introduction

The W3C is currently developing *RIF* (*Rule Interchange Format*) [6], a universal layer designed for exchanging rules between different and possibly heterogeneous systems over the Semantic Web. It is focused on the exchange more than on the development of a single system to fit all needs of all the already available rule systems, because it appears clear that a system which fits all needs is very difficult, if not impossible to build, due to the large syntactic and semantic differences between different systems or even in different modules of the same system. The RIF working group divided the language into *dialects* which are meant to be used in different situations, while maintaining the largest subset of rules in common. They are called *RIF profiles*: *Core*, *BLD* and *PRD*. While Core is formed by the base constructs of the language, BLD (Basic Logic Dialect) is focused on logic, while PRD (Production Rules Dialect) is based on the concept of production rules. Among other features, by treating F-Logic like frames equivalently to RDF triples, particularly the RIF Core and RIF BLD fragments, promise a standard format for publishing and exchanging rules on top of RDF.

* This work is partly funded by Science Foundation Ireland (SFI) project Lion-2 (SFI/08/CE/I1380) and an IRCSET postgraduate scholarship.

Likewise, ontologies in *OWL2RL*[10], a rule-based sublanguage of the Web ontology language *OWL2* [7], enables the support of inference over ontologies directly in rule-based system. This is achieved by giving a partial axiomatisation of the RDF OWL2 semantics in terms of first-order implications that can be encoded as rules.

At the moment few implementations of OWL2RL and RIF-Core exist since both languages are quite new. Moreover, we are not aware of any implementations – as of yet – that implement the combinations of RIF and OWL as standardised [2].

To fill this gap we propose and implemented a reduction of those languages to *DLVHEX* [4], a powerful disjunctive logic reasoner based on the Answer Set Programming paradigm. DLVHEX has it roots in *DLV*, a disjunctive Datalog system, but adds several features to the base language. The most interesting of them is the possibility to use natively higher order atoms and external atoms, which are added to the core language by means of a plug-in architecture. Through external atoms it is possible to inject procedural code in the otherwise purely declarative semantics of the language. This concept is very similar to libraries for other reasoners which enable interaction with external data sources, such as, e.g., the integration of RDF support in SWI-Prolog [13]. There already exist a rich collection of DLVHEX plugins for Semantic Web languages, such as SPARQL [11], RDF and OWL DL [5]. Our new plugin for RIF-Core and OWL2RL not only expands the interoperability of DLVHEX with these two new standards, but also enables the combination of both with the other data models and extensions, already accessible by plugins, for an evaluation, experiments and new applications by combining these languages with the expressive features of Answer Set Programming [1,3].

Our plugin allows DLVHEX to load and process RIF rule sets as well as OWL2RL ontologies. These are transformed to DLVHEX programs in a two-step translation: we first rewrite from OWL2RL to RIF-Core, and then perform a translation into DLVHEX. To this end there exist two different OWL2RL-to-RIF reduction methods, though, a static RIF rule set [12, Appendix 8.1] or dynamic a translation function from OWL2RL ontologies to RIF documents which yields RIF rules specifically to the input ontology [12, Appendix 8.2]. In comparison, the former approach bears some limitations in relation to interoperability with other RIF rule sets, and the combination of RIF with OWL ontologies as specified in [2] is rather based on the latter. Despite these restrictions, our current version of the OWL2RL reasoner transforms OWL2RL ontologies into RIF rules by the static rule set for the sake of a rapid first implementation. We will explain the limitations of this approach when doing it naively, and approximate the full dynamic combination of [2] by some extensions of the naive first translation.

In the following we give a description of our system, its current development status as well as accompanying examples in Section 2, and conclude with a report on our future plans in Section 3.

2 System Description

Our plugin consists of three parts: the OWL2RL to RIF-Core translation following [12], a RIF-Core to DLVHEX translator component, and the DLVHEX reasoner. In sequel we will provide more details to these components while we describe the system's workflow partitioned into its three essential stages:

Phase I - Translation from OWL2RL to RIF-Core. An OWL2RL ontology, given in RDF/XML, as input is forwarded to the OW2RL to RIF-Core translator which translates RDF triples of the input ontology to RIF frames and merges them with the static rule set from [12] to a RIF-Core document. The application of the static rule set to the RIF frames gained from the input will be performed during the evaluation of this RIF document later on.

Phase II - Reduction of RIF-Core to DLVHEX. The previously obtained RIF-Core document is preliminary reduced to a DLVHEX program. For that, the document is first parsed into an abstract syntax tree that is translated into a HEX program by a tree walking algorithm which gradually generates, adherent to a predefined set of translation rules, the corresponding HEX expressions from the visited tree nodes. This transformation includes reduction of features from RIF not directly expressible in our system to the processable input language of DLVHEX, e.g. Lloyd-Topor [8] transformation of rule bodies with discjunction, static type checking, or unnesting of external predicates, i.e. built-ins. Eventually, the generated program is forwarded to DLVHEX which returns a collection of answer sets.

Phase III - Answer Construction from DLVHEX to OWL2RL. Eventually, the answer sets, which are basically sets of ground facts, are simply transformed into a set of RIF ground atomic formulas.

Example – RIF to DLVHEX

The OW2RL to RIF-Core translation, executed in Phase I is straightforward. We give here only a small example for the RIF-Core to DLVHEX translation, occurring in Phase II. We apply it here to a test case from the RIF development group, `http://www.w3.org/2005/rules/wiki/Factorial_Forward_Chaining`:

```
Document( Prefix(pred
<http://www.w3.org/2007/rif-builtin-predicate#>) Prefix(func
<http://www.w3.org/2007/rif-builtin-function#>) Prefix(ex
<http://example.org/example#>) Group
  (
    ex:factorial(0 1)

    Forall ?N ?F? ?N1 ?F1 (
        ex:factorial(?N ?F) :-
            And(External(pred:numeric-greater-than-or-equal(?N1 0))
              ?N = External(func:numeric-add(?N1 1))
                   ex:factorial(?N1 ?F1)
                   ?F = External(func:numeric-multiply(?N ?F1)) )
)  )  )
```

This document describes the computation of the factorial for a positive integer n. Our DLVHEX plugin rewrites the above RIF document into the following two DLVHEX rules:

```
"ex:factorial"("0", "1") :-  . "ex:factorial"(VAR_N, VAR_F) :-
&pred_numeric_geq[VAR_N1, "0"](),
                   equal(VAR_N, VAR_extOutput_1),
                   &func_numeric_add[VAR_N1, "1"](VAR_extOutput_1),
                   "ex:factorial"(VAR_N1, VAR_F1),
                   equal(VAR_F,VAR_extOutput_2),
                   &func_numeric_multiply[VAR_N,VAR_F1](VAR_extOutput_2) .
```

The translation generates two rules, a fact and a *proper* rule, corresponding to the two input RIF rules. The universal quantifier of the second RIF rule is omitted here since DLVHEX rules are per se universal. RIF constants (CURIes, typed literals, quoted unicode strings, etc.), such as `ex:factorial` or `1`, are embraced by double quotes. Prefix names in curies will generally be expanded, but for better readability we didn't resolve them here. RIF built-in predicates and functions, such as `pred:numeric-greaterthan-or-equal` and `func:numeric-add`, are rewritten to an corresponding external DLVHEX atoms[1]. So far we support all RIF built-ins which may appear in a RIF document yielded by the OWL2RIF to RIF-Core translation. Beyond that, we also support all numeric predicates and functions implementable via calls to an XPath/XQuery Functions&Operators library. Besides, the lack of higher-order atoms in the resulting HEX program is no coincidence. In fact, those are not needed for a pure RIF-Core implementation. Our planned support for RIF-BLD as well as future RIF extensions similar to [5] will potentially demand higher-order features though.

Handling RIF-OWL2RL Combinations

The choice of a translation via the static rule set, applied in Phase I, seemed more convenient to implement at first view. since it supports a fast implementation. However, several limitations arise when translating OWL2RL into RIF via the static rules. Firstly, this method is rather inefficient compared to Reynolds's dynamic, pattern based approach [12, Appendix8.2], which creates more efficient RIF rules containing fewer free variables thus smaller grounding. Further and more problematic, the static rules as such are not suitable for RIF-OWL2RL combinations [2], i.e, a blend of OWL2RL rules with arbitrary RIF-Core rules. As pointed out in [2] the static rules create problems w.r.t. equality if applied to a RIF-OWL2RL combination, even if the RIF component is of RIF-Core. The reason lies in the possible introduction of equality through OWL2RL (via `[Object|Data]MaxCardinality` and `{Universe}FunctionalObjectProperty`) that can also affect the predicates existing in the RIF-Core component. In RIF-Core equality is only allowed in rule bodies and, thus, implications of equalities are not natively expressible. Likewise, our base system, DLVHEX, does not support equality natively, so we represent equality (which may only appear in rule bodies in RIF-Core) using `owl:sameAs`. This works out perfectly for the equality resembled by `owl:sameAs` on the level of RDF triples in the OWL2RL component [10, rules `eq-ref, eq-sym, eq-trans, eq-rep-s, eq-rep-p, eq-rep-o`], by axiomatisation in the OWL2RL rule set, but it is not comprehensively applicable in an analogous way to terms in RIF-Core, since arbitrary predicates or deeply nested external functions might occur in RIF rule sets which are unaffected by this axiomatisation.

Since we use the static rule set for the OWL2RL to RIF translation, at least for the time being, we developed a approximative rewriting for RIF rule sets for RIF-OWL2RL combination that allows us to catch these effects of equality. For a given RIF-OWL2RL combination $< R, G >$, where R is a RIF rule set and G is an RDF Graph, potentially

[1] Actually, for this particular example, we could have also exploited the built-in predicates of DLVHEX, which supports natively simple arithmetic functions such as sum, multiply and comparisons between variables. For the sake of the example, though, we decided to show how the systems can handle such external predicates and functions, in a simple way.

encoding an OWL2RL ontology, our algorithm runs through the following steps and outputs a rewritten RIF-Core program S:

1. Initialise S with R. Flatten all nestings of external predicates and functions in S by recursive substitution of nested terms with variables. For that, we need to express various equalities between arbitrary function terms. However, owl:sameAs is only applicable to express equality between simple terms. Thus, we need to introduce a new equality symbol '$\doteq$' which expresses equality between arbitrary terms. Since the value of each function term, by definition, belongs to an XML datatype we can think of $\doteq$ as equality as evaluated by XPath.[2]
2. Add the static RIF-Core rule set of Reynolds to S.
3. Add G in form of frame facts to S.
4. For any constant c that appears in R but not in G add the fact `c[owl:sameAs->c] .` to S.
5. For each rule of R in S rewrite any occurring atom $p(t_1, ..., t_n)$ where p is a constant and t_i is a simple RIF term ($1 \leq i \leq n$) to an atom $p(X_1, ..., X_n)$ where $X_i = t_i$ if t_i is a variable, else (i.e., t_i is a constant) X_i is a fresh variable.
6. Apply Lloyd-Topr rewriting for non-conjunctive rule bodies in S.
7. Optimisation by removing unnecessary `owl:sameAs` and $\doteq$ statements from the rule bodies in S.

Let us illustrate the effects of this algorithm by an example. Say R[3] contains

```
p(?x) :- Or( q(?x) r(?x,b) ) . r(c(2 * 2 + 2)). q(a). q(d) :- s(
1.3 + 0.7 ). s(1+1).
```

and $G = \{(\texttt{a}, \texttt{owl:sameAs}, \texttt{b})\}$. Then we get the following intermediate results for S:

After step 1:

```
p(?x) :- Or (q(?x)  r(?x,b) ) . r(c,?Y1) :- And( (?Y2 ≐ 2
* 2) (?Y1 ≐ ?Y2 + 2)   ). q(a). q(d) :- And( s( ?Y1 )
(?Y1 ≐ 1.3 + 0.7) ). s(?Y1) :- (?Y1 ≐ 1 + 1).
```

After step 2: $S := S \cup$ "Static Rule Set"

After step 3: $S := S \cup \{\texttt{a[owl:sameAs->b]}\}$

After step 4: $S := S \cup \{\texttt{c[owl:sameAs->c]}, \texttt{2[owl:sameAs->2]}\}$

After step 5:

```
p(?x) :- And ( Or (q(?x)  r(?x,?X1) ) ?X1[owl:sameAs->b] ) .
r(?X1,?Y1) :- And( ?X1[owl:samAs->c] (?Y2 ≐ 2 * 2) (?Y1
≐ ?Y2 + 2) ). q(a). q(?X1) :- And( ?X1[owl:samAs->d]  s(
?Y1 ) (?Y1 ≐ 1.3 + 0.7) ). s(?Y1) :- (?Y1 ≐ 1 +
1).
```

After step 7:

[2] In fact, on the stage of DLVHEX '$\doteq$' is evaluated by an external equality predicate implemented through XPath equality checks.

[3] Please note, that R deviates from the formal RIF syntax as we use here '$+$' and '$*$' for the built-in functions `func:numeric-add` and `func:numeric-multiply` in infix-notation for better readability.

```
p(?x) :- q(?x) . p(?x) :- And ( r(?x,?X1) ?X1[owl:sameAs->b] ) .
r(?X1,?Y1) :- And( ?X1[owl:samAs->c] (?Y2 ≐ 2 * 2) (?Y1
≐ ?Y2 + 2) ). q(a). q(?X1) :- And( ?X1[owl:samAs->d]  s(
?Y1 ) (?Y1 ≐ 1.3 + 0.7) ). s(?Y1) :- (?Y1 ≐ 1 +
1).
```

Our translation is realised as a plugin[4] to the DLVHEX system[5]. Furthermore, RIF-Core contains many built-ins in form of external predicates and functions. These external functions are computed by use of a standard XML Library that implements most of the common XPath/XQuery Functions& Operators [9]. At present, we support a subset of those, as we focused our attention on the built-ins which are mandatory for the reduction of OWL2RL reasoning to DLVHEX via RIF.

3 Conclusion and Future Work

We presented a DLVHEX plugin for OWL2RL and RIF-Core reasoning. The former is based on a 2-step reduction to DLVHEX via RIF-Core. This is, to our knowledge, the first attempt to implement RIF-OWL combinations a la [2], At our current stage of development we facilitate the translation to RIF by the static rule set of [12] which, as we have explained earlier, imposes restrictions on reasoning in combination with other RIF-Core documents. For the future we, therefore, will consider to modify the implementation of the first phase, switching from the static rule set to the dynamic rewriting by [12, Appendix8.2] similarly used in [2] which is based on RIF-BLD. Consequently, we will also try to extend the RIF-Core to DLVHEX translation in Phase II to more features of RIF-BLD. Moreover we plan to implement the remaining RIF built-ins to have a more complete translation from RIF-BLD to DLVHEX.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2002)
2. De Bruijn, J.: RIF rdf and OWL compatibility. Proposed recommendation, W3C (October 2009), `http://www.w3.org/TR/2010/PR-rif-rdf-owl-20100511/`
3. Eiter, T., Ianni, G., Polleres, A., Schindlauer, R.: Answer set programming for the semantic web. In: Tutorial at ESWC 2006 (June 2006)
4. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic-web reasoning. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 273–287. Springer, Heidelberg (2006)
5. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: KR (2004)
6. Kifer, M., Boley, H.: RIF basic logic dialect. Proposed recommendation, W3C (October 2009), `http://www.w3.org/TR/2010/PR-rif-bld-20100511/`

[4] For the source code and installation/usage instructions, please refer to `http://sourceforge.net/projects/dlvhex-semweb/` as well as `http://dlvhex-semweb.svn.sourceforge.net/viewvc/dlvhex-semweb/dlvhex-rifplugin/`

[5] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

7. Krötzsch, M., Patel-Schneider, P.F., Rudolph, S., Hitzler, P., Parsia, B.: OWL 2 web ontology language primer. Technical report, W3C (October 2009), http://www.w3.org/TR/2009/REC-owl2-primer-20091027/
8. Lloyd, J.W., Topor, R.W.: Making prolog more expressive. Journal of Logic Programming 1(3), 225–240 (1984)
9. Malhotra, A., Melton, J., Walsh, N.: XQuery 1.0 and XPath 2.0 functions and operators. Recommendation, W3C (January 2007), http://www.w3.org/TR/xpath-functions/
10. Motik, B., Fokoue, A., Horrocks, I., Wu, Z., Lutz, C., Cuenca Grau, B.: OWL 2 web ontology language profiles. W3C recommendation, W3C (October 2009), http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/
11. Polleres, A., Schindlauer, R.: dlvhex-sparql: A sparql-compliant query engine based on dlvhex. In: 2nd Int. Workshop on Applications of Logic Programming to the Web, Semantic Web and Web Services, ALPSWS 2007, pp. 332–347. Springer, Heidelberg (2007)
12. Reynolds, D.: OWL 2 RL in RIF. W3C working draft, W3C (October 2009), http://www.w3.org/TR/2009/WD-rif-owl-rl-20091001/
13. Wielemaker, J., Hildebrand, M., van Ossenbruggen, J.: Prolog as the fundament for applications on the semantic web. In: ALPSWS (2007)

A RPL through RDF: Expressive Navigation in RDF Graphs

Harald Zauner[1], Benedikt Linse[1,2], Tim Furche[1,3], and François Bry[1]

[1] Institute for Informatics, University of Munich, Oettingenstraße 67, 80538 München, Germany
[2] Thomson Reuters, Landsberger Straße 191a, 80687 München, Germany
[3] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, England

http://rpl.pms.ifi.lmu.de/

Abstract. RPL (pronounced "ripple") is the most expressive path language for navigating in RDF graphs proposed to date that can still be evaluated with polynomial combined complexity. RPL is a lean language well-suited for integration into RDF rule languages. This integration enables a limited form of recursion for traversing RDF paths of unknown length at almost no additional cost over conjunctive triple patterns.

We demonstrate the power, ease, and efficiency of RPL with two applications on top of the RPL Web interface. The demonstrator implements RPL by transformation to extended nested regular expressions (NREs). For these extended NREs we have implemented an evaluation algorithm with polynomial data complexity. To the best of our knowledge, this demo is the first implementation of NREs (or similarly expressive RDF path languages) with this complexity.

1 Motivation

With the promise of exciting "new kinds of usage scenarios", you finally got your boss at company C to embrace linked data and connect your community forum and contact database to other online communities and FOAF profiles of your contacts. Your boss now wants to put that technology to use: *"I want to cooperate with X on topic Y! Can you get me the name of any person that works at X and that's connected to us via people that are also interested in Y (so that they have an interest in connecting us). Oh, and none of the intermediates should be our competitor Z."*

Though the linked data movement and related initiatives like FOAF or SIOC provide specifically for this kind of scenario, most current analysis and query tools for RDF are not up to this task: SPARQL can only compute persons connected via fixed length paths due to the lack of any form of recursion. Under an (e.g., OWL-based) entailment regime that treats foaf:knows (the FOAF property used to build social networks) as a transitive property, SPARQL can compute all connected persons, but can not ensure that all intermediate persons share the same interest. The recent extension of SPARQL with property paths (to be incorporated into SPARQL 1.1) also fails at this task, as it only allows local restrictions on the traversed edges, but not on the traversed nodes, and no repetition (`*`) over paths with restrictions on nodes *and* edges.

Not only SPARQL fails at such analysis tasks on social networks or similarly interlinked data: Among the many RDF query and rule languages surveyed in [5], there is no language that can solve this analysis task and is not either impractical for large datasets (as NP- or Turing-complete languages such as SPARQLeR and TRIPLE) or only informally specified and now abandoned (as Versa). The more recent nSPARQL [8], an extension of SPARQL with nested regular expressions, can only solve parts of the above analysis task (but not the last sentence) and is not implemented in any publicly available tool.

In this demonstration, we show how to solve this and similar analysis tasks with a novel RDF path language, called RPL. The following RPL path expression solves our analysis problem (f being the FOAF namespace):

```
PATH [PATH _ <f:member C]
     (>f:knows [!PATH _ <f:member Z][PATH _ >f:interest _ >f:topic Y])*
      >f:knows [PATH _ <f:member X]
```

It returns all pairs of nodes such that the first node is an f:member of C and is connected to the second node via the specified path: It first traverses *outgoing* (indicated by `>`) f:knows edges and nodes that (1) are not members (employees) of Z and (2) have an f:interest edge that leads to some node that has Y as f:topic. It traverses arbitrarily many such edge-node pairs (indicated by `*`). The last node must also have an *incoming* (indicated by `<`) f:member edge from X.

Solving this analysis task in RPL is also *efficient* even on large RDF graphs: The demonstrated RPL implementation is, to the best of our knowledge, the first implementation of the bottom-up labeling algorithm for nested regular expressions from [8] on RDF data. It extends both nested regular expressions and the labeling algorithm with several important analysis features such as negation and regular expressions on literals and URIs. The extended labeling algorithm has been shown in [3] to have polynomial combined and data complexity.

With the analysis task solved, your job is save, your boss is impressed, and the Semantic Web vision is closer to reality.

2 A RPL through RDF Graphs

RPL is inspired by XPath, the dominant XML path language, in that it allows *nested predicates* on paths. Predicates allow a RPL user to express, in addition to *local* conditions on the path between two nodes, also *non-local* conditions on branches starting at a node on the path. Like XPath, RPL does not allow variables in path expressions and thus, all RPL queries are tree queries. RPL adapts XPath style path navigation to RDF and goes beyond XPath by replacing XPath's fixed closure axis with closure operators `?`, `*`, and `+` (as in [1] and [7]). RPL is set apart by three properties:

(1) RPL is designed from start off to be easily integrated into RDF rule and query languages such as XCERPT$^{\text{RDF}}$ [4] by allowing RPL expressions to appear in place of RDF predicates. Thus, a RPL expression e evaluates to a set of node pairs such that, between each pair of nodes, there is a path that matches e.

Together with the omission of variables, this ensures polynomial time and space data complexity (and polynomial combined) for RPL. This contrasts to most RDF path languages that either *(a)* allow the extraction of entire paths from RDF graphs (like SPARQLeR [6]) and have therefore exponential data complexity, or *(b)* allow variables to bind with nodes on the path (like PSPARQL [2]) again at the price of exponential combined complexity.

(2) RPL's rich syntax scales with the needs and abilities of the user: Beginners can describe paths only through the traversed edges (*edge-flavored* RPL) or only through the traversed nodes (*node-flavored* RPL), advanced users can place restrictions on both nodes and edges (*path-flavored* RPL). Together with RPL's predicates this is the main difference to the SPARQL property path extension that only allows restrictions on the edges traversed by a path.

(3) Expressive label tests with regular expressions and predicates with negation push RPL to the limits of RDF path languages with polynomial data complexity. E.g., the edge-flavored expression `EDGES >/.*train.*/*` traverses an arbitrary path whose forward-directed edges contain the keyword "train".

Predicates allow non-local restrictions and are arbitrary RPL expressions in square brackets. They can have either positive or negative sign (denoted by `!`).

3 System Description

Syntax and Semantics of RPL: To give a solid foundation for the discussion of the RPL implementation, we first briefly sketch its syntax:

$$\begin{array}{lll}
\langle \textit{rpl-expr} \rangle & ::= & \langle \textit{flavor} \rangle \ \langle \textit{adorned} \rangle + \\
\langle \textit{flavor} \rangle & ::= & \text{‘}\mathtt{EDGES}\text{’} \mid \text{‘}\mathtt{NODES}\text{’} \mid \text{‘}\mathtt{NODES<}\text{’} \mid \text{‘}\mathtt{NODES>}\text{’} \mid \text{‘}\mathtt{PATH}\text{’} \\
\langle \textit{adorned} \rangle & ::= & (\ \langle \textit{directed} \rangle \mid \text{‘}\mathtt{(}\text{’}\ \langle \textit{disjunctive} \rangle\ \text{‘}\mathtt{)}\text{’}\ )\ (\text{‘}\mathtt{?}\text{’} \mid \text{‘}\mathtt{*}\text{’} \mid \text{‘}\mathtt{+}\text{’})? \\
\langle \textit{directed} \rangle & ::= & (\text{‘}\mathtt{<}\text{’} \mid \text{‘}\mathtt{>}\text{’})?\ (\langle \textit{labeltest} \rangle \mid \langle \textit{predicates} \rangle) \\
\langle \textit{disjunctive} \rangle & ::= & \langle \textit{adorned} \rangle +\ (\text{‘}\mathtt{|}\text{’}\ \langle \textit{adorned} \rangle +\ )* \\
\langle \textit{predicates} \rangle & ::= & \text{‘}\mathtt{[}\text{’}\ \text{‘}\mathtt{!}\text{’}?\ \langle \textit{rpl-expr} \rangle\ (\ \text{‘}\mathtt{][}\text{’}\ \text{‘}\mathtt{!}\text{’}?\ \langle \textit{rpl-expr} \rangle\ )*\ \text{‘}\mathtt{]}\text{’} \\
\langle \textit{labeltest} \rangle & ::= & \text{‘}\mathtt{\_}\text{’} \mid \langle \textit{LITERAL} \rangle \mid \langle \textit{IRI\_REF} \rangle \mid \langle \textit{REGEXP} \rangle \\
 & \mid & \langle \textit{PN\_PREFIX} \rangle ?\ \text{‘}\mathtt{:}\text{’}\ (\langle \textit{PN\_LOCAL} \rangle \mid \langle \textit{REGEXP} \rangle)?
\end{array}$$

$\langle \textit{adorned} \rangle$ expressions form expression sequences and can be adorned by multiplicities, $\langle \textit{directed} \rangle$ expressions correspond to XPath node tests. We borrow most of the token classes from SPARQL (e.g. $\langle \textit{PN\_PREFIX} \rangle$ and $\langle \textit{PN\_LOCAL} \rangle$), but also allow regular expressions (enclosed in slashes) via $\langle \textit{REGEXP} \rangle$.

The semantics of RPL is specified by translation into ENREs, an extended version of nSPARQL's nested regular expressions (NREs) [8]. We have chosen this semantics to closely reflect our implementation that also translates RPL expressions into ENREs and gives the, to the best of our knowledge, first implementation of (E)NREs on RDF data.

Consider, e.g., the RPL expression `PATH` p `(>`t `_)+` that returns pairs of p together with any node reached from p over one or more intermediate nodes via t edges. This expression is translated into the ENRE

```
self_node::p/(next::t/self_node)+
```

ENREs have been tailored to fit the peculiarities of RPL and extend NREs from [8] by regular expressions for label tests, the negation of nested expressions (indicated by `!`), and three new navigation axes: `self_node` and `self_edge` act like `self` but only on nodes and edges, respectively; `next_or_next`$^{-1}$ is used for edges, where no direction is specified. Negation of nested expressions is realized as complement relation (i.e. $[\![$`[!`exp`]`$]\!]_G := \overline{[\![\texttt{[}exp\texttt{]}]\!]_G}$) and thus does not affect the polynomial complexity bounds.

Implementation: In the course of the demonstration, we demonstrate both RPL and how RPL queries are transformed into ENREs.

RPL is implemented in Java and uses Sesame 2.2.4 for accessing RDF data. We use the event based RDFHandler interface for fast and space efficient parsing of RDF triples into an in-memory graph representation (we choose this as it is an open question whether the bottom-up labeling algorithm used for evaluating ENREs and thus RPL can be implemented on a stream of RDF triples).

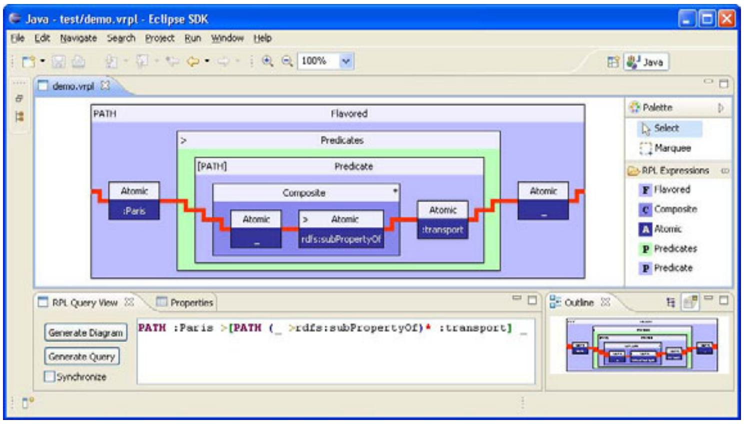
Through a number of normalization and verification steps, RPL queries are transformed into an equivalent ENRE. This ENRE is evaluated by an extended version of the bottom-up graph labeling algorithm from [8]. The algorithm recursively labels every node and edge v of the RDF graph with all nested expressions that v satisfies (i.e., there is a path beginning at v which satisfies the nested expression). The result is a product automaton $\mathcal{P} := \mathcal{G} \times \mathcal{A}$, where $\mathcal{G}$ is the RDF graph seen as an NFA (each node and each edge of the RDF graph is both an initial and final state, and the transitions are given by its triples), and $\mathcal{A}$ is the NFA that is induced by the ENRE seen as regular expression (Thompson's construction). $\mathcal{P}$ is used in a second phase to compute all node pairs (a, b), such that $(a, \cdot)$ is an initial state from which a final state $(b, \cdot)$ is reached in $\mathcal{P}$.

4 Riding RPL (Demo Description)

We demonstrate RPL using a Web-based, interactive interface: it provides a set of predefined application scenarios for a quick take-up of RPL. The application scenarios include the queried RDF graph, a visualization of that data, as well as a number of predefined queries that illustrate the strengths of RPL for analysing the respective data. The interface also allows users to enter their own RDF data (in any of the common RDF serializations Turtle, RDF/XML, N-Triples and N3) and their own RPL queries.

Furthermore, RPL queries can be comfortably authored in two separate Eclipse plugins: *visRPL* allows users to graphically compose RPL queries (see Figure 1) for new users, and another textual editor offers syntax highlighting and completion for more experienced users of RPL.

Application Scenario: Transportation Services. This application scenario is based on the nSPARQL [8] transportation services example. Both the data and the queries discussed here are available to the user as part of the Web-based interface. Figure 2 gives an impression of the data used in this scenario



```
PATH :Paris >[PATH (_ >rdfs:subPropertyOf)* :transport] _
```

Fig. 1. Visual Eclipse editor visRPL for RPL

(we abbreviate rdfs:subPropertyOf by rdfs:sp). The left hand of the graph shows a simple ontology for various connection types between cities, the right hand shows connections between a few European cities.

Imagine you are in Paris and want to find out which cities can be *directly* reached via any transportation service. You might start with a RPL query like `NODES> :Paris _` (or equivalent `PATH :Paris >_ _`), which will return all nodes that :Paris has an outgoing edge to. Hence, the pair (:Paris, :France) will also be part of the result set (due to the :country edge between them). To ensure that only transport edges are followed, we might be tempted to enumerate all types (train, bus, ferry) of transport edges in our data. Not only is such a solution clumsy, it also forces us to change our query whenever new types of transport edges are introduced. Thus, we use instead the RPL predicate that is shown in Figure 1. The query `PATH :Paris >[PATH (_ >rdfs:subPropertyOf)* :transport] _` specifies that we only follow edges from :Paris that are labeled as :transport or any of its sub-properties.

Once we submit this query, an AJAX request is sent to the RPL Web service which evaluates the RPL query and returns *(a)* a simplified and normalized, i.e. path-flavored RPL query, *(b)* an equivalent ENRE, and *(c)* the result of the

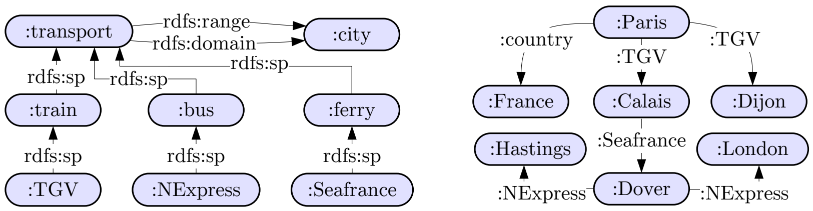
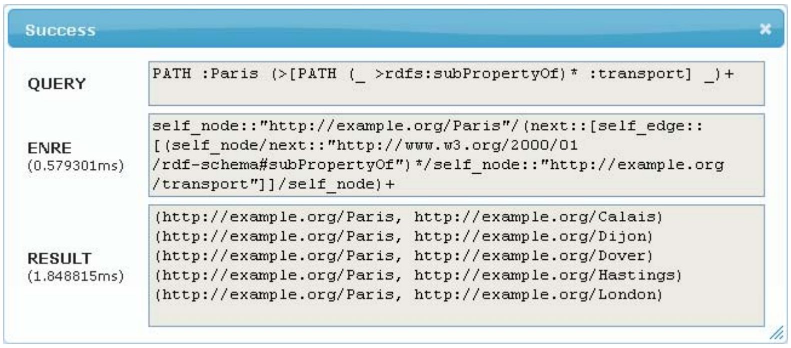


Fig. 2. Transportation services RDF graph


Success

QUERY
```
PATH :Paris (>[PATH (_ >rdfs:subPropertyOf)* :transport] _)+
```

ENRE (0.579301ms)
```
self_node::"http://example.org/Paris"/(next::[self_edge::
[(self_node/next::"http://www.w3.org/2000/01
/rdf-schema#subPropertyOf")*/self_node::"http://example.org
/transport"]]/self_node)+
```

RESULT (1.848815ms)
```
(http://example.org/Paris, http://example.org/Calais)
(http://example.org/Paris, http://example.org/Dijon)
(http://example.org/Paris, http://example.org/Dover)
(http://example.org/Paris, http://example.org/Hastings)
(http://example.org/Paris, http://example.org/London)
```

Fig. 3. Result dialog: RPL query, ENRE, result

evaluation (or an error message). The Web interface displays this information together with timing information in a dialog as shown in Figure 3.

It turns out that none of the directly reachable cities interests us today. Thus, we decide that intermediate stops are acceptable and want to adapt our query accordingly. We simply have to add a closure multiplicity (+) to get to any place that is reachable by one or more transport edges from :Paris, see Figure 3.

Unfortunately, we get really sick when traveling with a ferry and thus would like to exclude connections that use a ferry. Again, the modification is straightforward: we add another predicate (! indicates negation) that does not allow :ferry edges (and their sub-properties) to be traversed. The adapted query is

```
PATH :Paris (>[PATH (_ >rdfs:subPropertyOf)* :transport][
               !PATH (_ >rdfs:subPropertyOf)* :ferry] _)+
```

It evaluates to the set {(:Paris, :Calais), (:Paris, :Dijon)}. The cities :Dover, as well as :London and :Hastings (which are only reachable over :Dover), are excluded now as the only transport link from Calais to Dover is a kind of ferry.

Finally, RPL is also able to express a relevant part of the RDFS entailment rules: the following query retrieves all nodes that "are" cities according to RDFS (i.e. all nodes that have an rdf:type edge to :city in the RDFS closure of Fig. 2).

```
NODES ( [PATH _ >rdf:type (_ >rdfs:sc)* :city]
      | [EDGES >[PATH (_ >rdfs:sp)* _ >rdfs:domain (_
          >rdfs:sc)* :city]]
      | [EDGES <[PATH (_ >rdfs:sp)* _ >rdfs:range (_
          >rdfs:sc)* :city]])
```

Acknowledgements. The research leading to these results has received funding from the *European Research Council* under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 246858.

References

1. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.: The Lorel query language for semistructured data. Int. J. on Digital Libraries 1(1) (1997)
2. Alkhateeba, F., Baget, J.-F., Euzenat, J.: Extending SPARQL with regular expression patterns. J. of Web Semantics (2009)
3. Bry, F., Furche, T., Linse, B.: The perfect match: RPL and RDF rule languages. In: RR (2009)
4. Bry, F., Furche, T., Linse, B., Pohl, A., Weinzierl, A., Yestekhina, O.: Four lessons in versatility or how query languages adapt to the web. In: Bry, F., Maluszynski, J. (eds.) Semantic Techniques for the Web, The Rewerse Perspective. LNCS, vol. 5500, pp. 50–160. Springer, Heidelberg (2009)
5. Furche, T., Linse, B., Bry, F., Plexousakis, D., Gottlob, G.: RDF querying: Lang. constructs and eval. methods compared. In: Barahona, P., Bry, F., Franconi, E., Henze, N., Sattler, U. (eds.) Reasoning Web 2006. LNCS, vol. 4126, pp. 1–52. Springer, Heidelberg (2006)
6. Kochut, K., Janik, M.: SPARQLeR: Extended SPARQL for semantic association discovery. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 145–159. Springer, Heidelberg (2007)
7. Marx, M.: Conditional XPath. ACM Trans. on Database Sys. (TODS) 30(4) (2005)
8. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: A navigational language for RDF. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 66–81. Springer, Heidelberg (2008)

Author Index