Diego Calvanese
Georg Lausen (Eds.)

LNCS 5341

Web Reasoning and Rule Systems

Second International Conference, RR 2008
Karlsruhe, Germany, October/November 2008
Proceedings

RR2008

Springer

Diego Calvanese   Georg Lausen (Eds.)

Web Reasoning and Rule Systems

Second International Conference, RR 2008
Karlsruhe, Germany, October 31 - November 1, 2008
Proceedings

Springer

Volume Editors

Diego Calvanese
Free University of Bozen-Bolzano, Faculty of Computer Science
Piazza Domenicani 3, 39100 Bolzano, Italy
E-mail: calvanese@inf.unibz.it

Georg Lausen
Albert-Ludwigs-Universität Freiburg
Fakultät für Angewandte Wissenschaften, Institut für Informatik
Georges-Köhler-Allee, Gebäude 51, 79110 Freiburg, Germany
E-mail: lausen@informatik.uni-freiburg.de

Library of Congress Control Number: 2008937462

CR Subject Classification (1998): D.2, H.2.4, I.2.4, H.3.5, H.5

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

ISSN 0302-9743
ISBN-10 3-540-88736-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-88736-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12546060 06/3180 5 4 3 2 1 0

Preface

This volume contains the proceedings of the Second International Conference on Web Reasoning and Rule Systems (RR2008), which was held on October 31 and November 1 in Karlsruhe, Germany.

The International Conference on Web Reasoning and Rule Systems (RR) is the major forum for discussion and dissemination of new results on all topics concerning web reasoning and rule systems. RR2008 built on the success of the First International Conference on Web Reasoning and Rule Systems RR2007, which received enthusiastic support from the Web Rules community. In 2008, as documented by this proceedings volume, RR continued the excellence of the new series.

The reasoning landscape features theoretical areas such as knowledge representation (KR) and algorithms; design aspects of rule markup; design of ontology languages; engineering of engines, translators, and other tools; efficiency considerations and benchmarking; standardization efforts, such as the Rules Interchange Format activity at W3C; and applications. Of particular interest has been the use of rules to facilitate ontology modeling, and the relationships and possible interactions between rules and ontology languages like RDF and OWL, as well as ontology reasoning related to RDF and OWL, or querying with SPARQL.

We received 35 submissions, each of which was reviewed by at least 3 Program Committee members. The committee decided to accept 21 papers, among these 12 full papers, 4 short papers and 5 poster presentations. The program also featured two invited talks, one by Michael Kifer on the standardization activity around RIF, and one by Boris Motik on the theoretical foundations underlying the integration of description logics and rules. The talk by Michael Kifer was also broadcast as a joint keynote to the International RuleML Symposium on Rule Interchange and Applications (RuleML2008), held in parallel in Orlando (USA).

We would like to thank all the people who invested their time and efforts and made the organization of this conference possible. The General Chair Thomas Eiter, and the members of the RR Steering Committee, who provided valuable advice when critical decisions had to be taken, the members of the Program Committee and the additional reviewers, who helped in shaping a challenging conference program, and Sebastian Rudolph who took care of the local requirements for running the conference. Finally, we would like to thank Andrei Voronkov, for providing us with the EasyChair system, which greatly simplified the handling of all stages from the paper submission to the production of these proceedings.

August 2008

Diego Calvanese
Georg Lausen

Organization

General Chair

Thomas Eiter	Technical University of Vienna, Austria

Program Chairs

Diego Calvanese	Free University of Bozen-Bolzano, Italy
Georg Lausen	University of Freiburg, Germany

Local Chair

Sebastian Rudolph	University of Karlsruhe, Germany

Steering Committee

Harold Boley	University of New Brunswick, Canada
Francois Bry	Ludwig-Maximilians-Universität München, Germany
Thomas Eiter	Technische Universität Wien, Austria
Pascal Hitzler	University of Karlsruhe, Germany
Michael Kifer	State University of New York, USA
Massimo Marchiori	Università di Padova, Italy
Jeff Z. Pan	University of Aberdeen, UK
Riccardo Rosati	Sapienza Università di Roma, Italy
Christian de Sainte Marie	ILOG, France
Bruce Spencer	University of New Brunswick, Canada
Holger Wache	University of Applied Sciences NW Switzerland, Switzerland

Program Committee

Jose Alferes	Universidade Nova de Lisboa, Portugal
Jürgen Angele	Ontoprise GmbH, Karlsruhe, Germany
Marcelo Arenas	Pontificia Universidad Catolico de Chile, Chile
Uwe Assmann	Technische Universität Dresden, Germany
Chitta Baral	Arizona State University, USA
Leopoldo Bertossi	Carleton University, Ottawa, Canada
Jos de Bruijn	Free University of Bozen-Bolzano, Italy
Christoph Bussler	BEA Systems, Inc. USA

Additional Reviewers

Andreas Bartho, Meghyn Bienvenu, Alexandros Chortaras, Jianfeng Du, Anna Lisa Gentile, Jonathan Goldstein, Jakob Henriksson, Matthias Knorr, Yue Ma, Alessandra Martello, Tobias H. Naeth, Anne Schlicht, David Thau, Sebastian Wandelt

Table of Contents

Rule Interchange Format: The Framework

Michael Kifer

State University of New York at Stony Brook, USA

Abstract. The Rule Interchange Format (RIF) is an activity within the World Wide Web Consortium aimed at developing a Web standard for exchanging rules. The need for rule-based information processing on the Semantic Web has been felt ever since RDF was introduced in the late 90's. As ontology development picked up pace this decade and as the limitations of OWL became apparent, rules were firmly put back on the agenda. RIF is therefore a major opportunity for the introduction of rule based technologies into the main stream of knowledge representation and information processing on the Web.

Despite its humble name, RIF is not just a format and is not primarily about syntax. It is an extensible framework for rule-based languages, called RIF *dialects*, which includes precise and formal specification of the syntax, semantics, and XML serialization. In this paper we will discuss the main principles behind RIF, introduce the RIF extensibility framework, and outline the Basic Logic Dialect—the only fully developed RIF dialect so far.

1 Introduction

The *Rule Interchange Format* (RIF) activity within the World Wide Web Consortium (W3C) aims to develop a standard for exchanging rules among disparate systems, especially on the Semantic Web. Given that the existing rule systems, both commercial and research prototypes, have wide variety of features and differ not only syntactically but also—more importantly—semantically, the goal of the RIF effort is not at all simple. Some systems extend one another syntactically and/or semantically, but in many cases this is true only to a degree. Other rule systems are largely incompatible, each having features that the other system does not. With this diversity, how can interoperability be achieved?

The vision of RIF is a collection of *dialects*—an extensible set of languages with rigorously defined syntax and semantics. Extensibility here means that new dialects can be added, if sufficient interest exists, and the languages are supposed to share much of the syntactic and semantic apparatus.

Because of the emphasis on rigor and semantics, the term "format" in the name of RIF might seem a misnomer. However, making a cute acronym is not the only reason for the choice of this term. The idea behind rule exchange through RIF is that the different systems will be able to map their languages (or substantial parts thereof) to and from the appropriate RIF dialects in *semantics-preserving* ways and thus rule sets and data could be communicated by one system to another provided that the systems can find a suitable dialect, which they both support. The intermediate RIF language is supposed to be in the XML format, whence the term "format" in the RIF name.

The RIF Working Group has plans to develop two kinds of dialects: logic-based dialects and dialects for rules with actions. The logic-based dialects include languages based on first-order logic and also a variety of logic programming dialects based on the different non-first-order semantics such as the well-founded and stable semantics [22,12]. The rules-with-actions dialects will be designed for production rule systems, such as Jess and Drools [15,14], and for reactive rules such as those represented by XChange [4], FLORA-2 [16], and Prova [18]. At the time of this writing, only the *Basic Logic Dialect*, RIF-BLD (which belongs to the first category), has been substantially completed and is in the "Last Call" status in the W3C standardization process [2]. In the second category, a *Production Rule Dialect*, RIF-PRD, is under active development [8].

These plans are very ambitious, and in the beginning it was not at all obvious how the different dialects could be made to substantially share syntactic and, especially, semantic machinery. Even within the logic-based category the dialects are expected to have vastly different semantics: the first-order semantics warrants inferences that are different from those warranted by the logic programming semantics, and the various logic programming semantics do not agree in all cases. This is where the RIF extensibility framework comes in. At present, only the *Framework for Logic Dialects*, RIF-FLD, has been worked out to sufficient degree of detail [3], and this is the main subject of this paper.

We assume general familiarity with first-order logic syntax and semantics, and with the idea of rule-based languages, especially logic programming languages like Prolog [7]. We also assume that the reader understands the difference between first-order logic based languages and those based on logic programming; especially the difference that the notion of *negation* plays in both.

This survey is organized as follows. In Section 2, we give an overview of the RIF framework for logic dialects. Section 3 describes the syntactic machinery of the framework, including the notions of terms, formulas, signatures, and well-formedness. Section 4 describes the semantic framework. In Section 5 we give a brief introduction to the Basic Logic Dialect of RIF and show how it can be described in terms of the RIF framework. Section 6 concludes the paper.

2 RIF Framework for Logic Dialects—An Overview

The RIF *Framework for Logic Dialects*, RIF-FLD, is a formalism for specifying all logic dialects of RIF, including the Basic Logic Dialect [1]. In itself, FLD is a logic in which both the syntax and semantics of the dialects are described through a number of mechanisms that are commonly used in practice and in literature, but are rarely brought all together. Fusion of all these mechanisms is required because the framework must be broad enough to accommodate several different types of logic languages and because various advanced mechanisms are needed to facilitate translation into a common framework. RIF-FLD gives precise definitions to these mechanisms, but allows details to vary. The design of RIF envisages that its future logic dialects will be based on RIF-FLD and will be defined as specializations of FLD. Being all derived from the same framework will ensure that RIF dialects are syntactically and semantically compatible in the sense that extensions, restrictions, and common subsets of the different dialects will be formally identifiable and rule systems would be able to communicate their rule sets using a collection of such dialects.

The framework has three main components: the syntactic framework, the semantic framework, and the XML framework. The syntactic framework defines a general mechanism for specifying which kinds of terms and formulas are allowed and how to specialize this mechanism to produce specific dialects. The semantic framework provides model-theoretic mechanisms for specifying how logical inference is to be defined in the derived dialects. The XML framework defines the general principles for mapping the syntax of RIF-FLD to a concrete XML interchange format.

As an example of this approach, the RIF Basic Logic Dialect is normatively defined as a specialization of RIF-FLD. Having RIF-FLD is a major advantage because the specification of RIF-BLD as a specialization of RIF-FLD is very short and easy to grasp. For comparison, RIF-BLD is also specified directly, without relying on the framework. This specialization is also normative, but much longer and more complex. It is required that the two specifications of BLD are equivalent and any discrepancy must be treated as a mistake to be corrected.

In the following sections, we will provide an informal survey of the syntactic and semantic frameworks. It is informal both in order to be brief and also because the reader is encouraged to consult the definitive document [3].

3 The Syntactic Framework

The syntactic framework defines the types of terms and formulas that are allowed in a dialect. A specific dialect might choose to restrict certain combinations of symbols and throw out some combinations altogether.

3.1 Terms: The Object Level

The framework defines the following types of terms (among others: it is not the purpose this this survey to complete):

– *Constants and variables.* In the RIF presentation syntax, variables are denoted using alphanumeric symbols prefixed with a "?"-mark, and we will also do so here.
– *Positional terms.* If t and t_1, ..., t_n are terms then $t(t_1\ ...\ t_n)$ is a positional term.[1] These are like the usual terms of first-order logic except that

[1] The presentation syntax of RIF does not use commas to separate the arguments in predicates and terms. It is an abstract syntax and, as such, it omits certain details that might be important for unambiguous parsing.

the symbols are not necessarily partitioned into individuals, functions, and predicates (any such restrictions are left to the dialects' discretion). In addition, variables are allowed to occur anywhere a term can. Thus, a positional term can be as general as a HiLog term [5] and expressions of the form $?X(abc\ ?W)(?Y\ ?Z(?V\ 33))$ are well within the limits of what is allowed.

– *Terms with named arguments.* The arguments of a term can be named as in $person(name{\rightarrow}Bob\ age{\rightarrow}33)$. Such a term is distinct from, say, $person(Bob\ 33)$ or $person(spouse{\rightarrow}Bob\ age{\rightarrow}33)$. However, the *order* of the named arguments within such a term is immaterial, so $person(name{\rightarrow}Bob\ age{\rightarrow}33)$ and $person(age{\rightarrow}33\ name{\rightarrow}Bob)$ are indistinguishable.
– *Frame and classification terms.* RIF-FLD includes certain terms borrowed from F-logic [17]. A *frame* term has the form $t[p_1{\rightarrow}v_1\ ...\ p_n{\rightarrow}v_n]$, where t, p_1, ..., p_n, v_1, ..., v_n are terms. The order of the attribute specifications (the $p_i{\rightarrow}v_i$'s) is immaterial, like in the case of terms with named arguments. However, frames have very different semantics compared to the named argument terms. For instance, in $bob[name{\rightarrow}Bob\ age{\rightarrow}33]$, bob denotes an object and $name{\rightarrow}Bob$, $age{\rightarrow}33$ are statements about the properties of that object. In contrast, $person(name{\rightarrow}Bob\ age{\rightarrow}33)$ is not a statement about the object *person*. Here *person* is the name of a relation type (think of a database table) and $name{\rightarrow}Bob\ age{\rightarrow}33$ describes a particular relation of that type. So, $bob[spouse{\rightarrow}mary]$ would still be a statement about the same object *bob* (just about some other of its properties), while the statement $person(spouse{\rightarrow}Mary)$ would have no relationship to the earlier statement $person(name{\rightarrow}Bob\ age{\rightarrow}33)$.

  Classification terms include *membership* and *subclass* terms. Here $t\#s$ represents a membership relationship between the member-object t and the class-object s; $s\#\#c$ is a term that represents the subclass relationship between the objects s and c.[2] For instance, $student\#\#person$.
– Other kinds of terms include equality and external terms. The latter represent references to outside sources of information and built-ins.

Since communication between the different rule systems through the medium of RIF is supposed to be by translation, one might ask why so many different kinds of terms? After all, it is well known that everything can be encoded using just the first-order terms; in fact lists alone suffice. The answer is *model-preservation* or *round-tripping.* One of the requirements in RIF is to support round-tripping, i.e., the ability to translate a rule set from, say, system S_1 to RIF, then to S_2, then back to RIF, back to S_1, and get not only a semantically equivalent set of rules, but essentially the same set of rules from the modeling points of view. What this is supposed to mean precisely has not been addressed, but the intuitive idea is that if something was modeled as an object (a frame term) then it should stay an object and not metamorphose itself into a relation (a positional or a named argument term) after returning back to S_1. Likewise, the subclass and

[2] Those familiar with F-logic might be surprised to see $t\#s$ and $s\#\#c$ instead of $t : s$ and $s :: c$, but the colon has been irrevocably appropriated for other purposes in the world of W3C standards.

membership relationships are well-established modeling primitives and must be recognized by the syntax. This also simplifies translation to and from RIF, and makes it more natural.

The other question that comes to mind is why things that are normally called formulas (e.g., frames and classification terms in F-logic [17]) are called *terms* in RIF-FLD? The answer is that RIF-FLD is required to support a degree of *reification*—the ability to represent formulas (which are statements about true facts or beliefs) as terms (i.e., objects). In this way, RIF will allow dialects in which statements can be made about other statements, and these *meta*-statements can then be processed by rules.

3.2 Formulas: The Statement Level

The logic RIF framework defines several types of formulas, most of which are adaptations from other known logics. However, in RIF-FLD they are all together in one logic system.

- *Atomic formulas*: A term is also an atomic formula. Like in HiLog [5], this blurs the distinction between objects and statements about objects and lays a foundation of the infrastructure for meta-reasoning in RIF dialects that might choose to support it.
- *Conjunction and disjunction*: These are the usual connectives in first-order logic. The RIF syntax for that is `And`(ϕ_1 ... ϕ_n) and `Or`(ϕ_1 ... ϕ_n).
- *Negation*: RIF-FLD supplies both the classical negation as used in first-order logic, denoted `Neg`, and a symbol for default negation, as used in logic programming. The latter is intended for logical notions of default negation, such as those based on the well-founded and the stable-model semantics [22,12]—*not* for negation-as-failure, as used in Prolog [6]. In view of this, the current choice of the symbol for default negation, `Naf`, is misleading and might be replaced in the future. It is also possible that *explicit negation* (a weaker form of classical negation that is sometimes used in logic programming [13]) might be added in the future.[3]
- *Rule implication*: A rule implication is a formula of the form *phi* `:-` ψ. This is the notion of implication as used in logic programming; it is different from the classical implication and is *not* equivalent to `Or`(ϕ `Neg` ψ).
- *Quantification*: A quantified formula is, as usual, a formula of the form `Forall` $?V_1 \ldots ?V_n$ (ϕ) or `Exists` $?V_1 \ldots ?V_n$ (ϕ).

Apart from these, FLD also has `Group`-formulas and `Document`-formulas. A group formula is simply a set of formulas of the above form, and groups can be nested. This type of formulas exists just for convenience and for possible future enhancements. One convenience is the ability to assign an identifier (say, a URL) and meta-data to a group of formulas. This information can then be used in other Web documents.

[3] Since true classical negation and explicit negation are never used together, it is also possible that `Neg` will be used for both.

A `Document`-formula generalizes what we earlier informally called a "rule set." The Web consists of documents and this is also a structural unit chosen for RIF. An important aspect of documents is that one can *import* the other. This provides a degree of modularity similar to what exists in other Web standards, such as XML Schema and OWL [11,9].

Documents also provide a convenient way to localize constant symbols to particular documents and avoid clashes. This is particularly important for logic programming languages where it is common to use intermediate predicates that are not supposed to have meaning outside of a particular document.

3.3 Signatures: The Key to Extensibility

One of the most important ingredients that makes RIF-FLD into a *framework* for defining other languages (dialects) is the concept of a signature. Signatures determine which terms and formulas are *well-formed*. It is a generalization of the notion of a *sort* in classical first-order logic [10]. Each symbol has an associated signature. A signature defines, in a precise way, the syntactic contexts in which the symbol is allowed to occur.

For instance, the signature associated with a symbol p might allow p to appear in a term of the form *f(p)*, but disallow it to occur in the term *p(a,b)*. The signature for *f*, on the other hand, might allow that symbol to appear in *f(p)* and *f(p,q)*, but disallow *f(p,q,r)* and *f(f)*. Note that, say, *f(f)* is still a term according to our earlier definition; it is just not a *well-formed* term. In this way, it is possible to control which symbols are used for predicates and which for functions, where variables are allowed to occur and where they are not allowed.

A *signature* is a statement of the form $\eta\{e_1,\ ...,\ e_n,\ ...\}$ where η is the name of the signature and $\{e_1,\ ...,\ e_n,\ ...\}$ is a countable set of *arrow expressions*. The number of such expressions in a particular signature can be zero or more, or it can be infinite. The dialects decide for themselves. In RIF-BLD, signatures can have at most one arrow expression. Dialects that support polymorphism may allow more than one arrow expression in a signature. HiLog [5], for example, puts a countably infinite number of arrow expressions in all signatures.

An *arrow expression* is a statement of the form $(\kappa_1\ ...\ \kappa_n) \Rightarrow \kappa$, where κ, κ_1, ..., κ_n are signature names. For instance, if *term* is a signature name then $(\,) \Rightarrow \mathit{term}$ and $(\mathit{term}) \Rightarrow \mathit{term}$ are signatures.

There is more to the notion of arrow expression that the above suggests. For instance, the above are arrow expressions for just the *positional* terms. There are also signatures for terms with named arguments, frames, and signatures can be organized into class hierarchies. However, we will ignore these aspects and focus on the essentials.

Signatures are used to control the context in which symbols occur using the notion of *well-formedness*. Earlier we defined the notion of terms and formulas, but those definitions do not say whether a term or a formula is well-formed. In order to define this notion we must assume that every symbol in the alphabet of the language is assigned a unique signature. How exactly this is done depends on a dialect. For instance, BLD imposes very strict conditions on signatures, which makes it possible to assign signatures by the context in which the symbols are used. Terms are well-formed if their structure conforms to the following rules.

– A constant or variable symbol with signature η is a well-formed term with signature η.
– A term $t(t_1 \ldots t_n)$ is well-formed and has a signature σ if and only if
  - `t` is a well-formed term that has a signature that contains an arrow expression of the form $(\sigma_1 \ldots \sigma_n) \Rightarrow \sigma$; and
  - Each t_i is a well-formed term with signature σ_i.

This is not a full definition. It omits terms with named arguments, frames, membership and subclass terms, and other aspects. The full definition can be found in [3]. However, this partial definition should convey the idea. For instance, if p has the signature $mysig\{(obj) \Rightarrow obj,\ (obj\ obj) \Rightarrow obj,\ (obj\ obj\ obj) \Rightarrow obj\}$ and a, b, c each has the signature $obj\{\,\}$ then $p(p(a)\ p(a\ b\ c))$ is a well-formed term with signature $obj\{\,\}$. On the other hand, $p(a\ b\ c\ a)$ is a term, but not a well-formed one, since the signature of p has no arrow expression that permits p to have four arguments. The following is an even more telling example. Suppose *John* and *Mary* are symbols with the signature $obj\{\,\}$, the variable *?P* has the signature $h_2\{(obj\ obj) \Rightarrow obj\}$, and *closure* has the signature $h_3\{(h_2) \Rightarrow p_2\}$, where p_2 is the name of the signature $p_2\{(obj\ obj) \Rightarrow obj\}$. Then $?P(John\ Mary)$ and $closure(?P)(John\ Mary)$ are well-formed terms with signature $obj\{\,\}$.

Designers of each particular RIF dialect can decide which signatures can be assigned to which symbols and in this way fully determine the syntax of the dialect. Thus, RIF-FLD provides a general framework, which dialects can use to specify their syntaxes. The present draft of RIF-BLD uses a different technique for defining well-formed formulas, but a future draft will extend signatures to cover well-formedness of formulas by assigning signatures to logical connectives. In particular, RIF dialects would be entitled to introduce connectives, such as modal operators, which do not explicitly exist in RIF-FLD.

4 The Semantic Framework

The RIF-FLD semantic framework defines the notions of semantic structures and of models for RIF-FLD formulas. The semantics of a dialect is derived from these notions by specializing the following parameters.

1. The effect of the syntax.
   The syntax of a dialect may limit the kinds of terms that are allowed. For instance, if a dialect's syntax excludes frames or terms with named arguments then the parts of the semantic structures whose purpose is to interpret those types of terms become redundant.
2. Truth values.
   The semantic framework allows formulas to have truth values from an arbitrary partially ordered set of truth values, TV. A concrete dialect must

select a concrete partially or totally ordered set of truth values. For instance, most dialects are expected to stay within the regular two-valued category, but, for example, logic programming dialects that are based on the well-founded semantics would use a three-valued logic where the order is $true > undefined > false$.

3. Datatypes.
   A datatype is a set of symbols that have a fixed interpretation in any semantic structure. RIF-FLD defines a set of core datatypes that each dialect is required to include as part of its syntax and semantics. However, it does not limit dialects to just the core types: they can introduce additional datatypes, and each dialect must define the exact set of datatypes that it includes.

   This is just a remark in passing about the role of datatypes in RIF, which is beyond the scope of this survey. RIF datatypes are defined in a separate document produced by the working group [20].
4. Logical entailment.
   Logical entailment in RIF-FLD is defined with respect to an unspecified set of *intended models.* A RIF dialect must define which models are considered to be intended. For instance, one dialect might specify that *all* models are intended (which leads to classical first-order entailment), another may regard only the minimal models as intended, while a third might use only well-founded or stable models [22,12].

We will not reproduce all the definitions here, but instead will highlight the most interesting aspects. The definition of semantic structures is pretty standard, especially to those who are familiar with F-logic and HiLog [17,5]. The main differences are the mechanisms for dealing with multiple truth values (recall that the set TV of truth values can include more than the standard *true* and *false*) and formula reification. It amalgamates the techniques from [17,5] to allow reification of frames.

Another interesting technique is used to define the semantics of document formulas. Recall that documents can import other documents, and documents can have local symbols. So, import is not just a mechanical union of all the imported document: the local symbols need to be disambiguated. FLD provides a model-theoretic semantics for that.

What makes FLD into a true framework for a range of different semantics is the concept of *entailment* that is based on the notion of intended models. To make the problem clear and highlight the difficulties, let us recall that apart from the syntax, what makes the different logic languages really different is their notion of entailment, i.e., the way they determine which formulas are regarded as consequences of other formulas. For instance, a large subset of first-order logic can be seen as a rule-based language. In such a language, the formula $p \leftarrow \neg p$ logically entails p, but not, say, q. If the same formula is considered to be part of a logic programming language with $\neg$ understood as default negation then the situation is different. First, there are several semantics for default negation, and two of them are widely used. According to the stable model semantics [12], $p \leftarrow \neg p$ is an inconsistent formula, so every conclusion follows, including q. According to the other popular semantics, the well-founded semantics [22], $p \leftarrow \neg p$ is consistent, but *nothing* of interest follows from it: neither p nor q.

The question therefore is: how does one accommodate all these different semantics in one framework so that the different RIF dialects could share the same machinery and be compatible with each other? The solution adopted in RIF-FLD was proposed by Shoham over two decades ago [21] when he observed that many logics that seemingly use completely different notions of entailment share essentially the same elements and can be explained away with the help of one simple definition.

We already talked about the notion of semantic structures, which is also often called *interpretation* in the literature. The purpose of semantic structures is to define certain sets and functions, which together determine the truth value (drawn from the set TV) of every well-formed formula in the logic language. If a semantic structure assigns the value *true* to a formula then it is said to be a *model* of that formula.

If **S** is a set of semantic structures then we say that one formula, ϕ, **S**-entails another formula, ψ, if and only if for every semantic structure in **S**, if it is an *intended* model of ϕ then it is also a model of ψ.

It turns out that all the interesting logic-based rule languages, including first-order logic and many others, define their notions of entailment in this or an equivalent way. The only difference is the set **S**, which they consider in defining entailment, and what they consider to be an "intended" model. For instance, first-order logic has the simplest definition in this regard: **S** is just the set of all semantic structures and every model is intended. Other logics are more picky. For instance, **S** might contain only Herbrand semantic structures [19], and only minimal (in a certain sense) models might be considered as intended. Yet other languages have their own ideas about what is intended. We already mentioned the well-founded semantics and the stable-model semantics, for which the intended models are, as their names suggest, the well-founded models and the stable models, respectively [22,12].

So, the bottom line is that RIF-FLD defines entailment with respect to the sets of intended models, as above, but it does not specify what these intended models are—it only defines semantic structures in general. It is left to the dialects to choose the appropriate notion.

5 The Basic Logic Dialect

The Basic Logic Dialect, RIF-BLD, is currently the only fully specified dialect of RIF. From the expressivity point of view, this dialect corresponds to the familiar Horn subset of logic programming [19]. No negation of any kind is allowed in the rule head and in the body. However, RIF-BLD has many syntactic extensions with respect to stock Horn rules. These include:

- Conjunctions in rule heads and disjunctions in rule bodies.
- Frames, membership, and subclass formulas.
- Predicates and functions with named arguments.

– Data types, group and document formulas.
– Equality both in rule heads and bodies.

There is also one notable restriction compared to FLD (and to many logic programming languages, like Prolog): as in a standard textbook version of first-order logic, every symbol is allowed to occur in at most one context in any document (including the imported documents). Thus, if a symbol occurs in the context of, say, binary predicate then it cannot occur as a ternary predicate. It cannot also occur as a function symbol or individual constant.

Ostensibly, these extensions and restrictions are supposed to simplify round-trippable translations to and from RIF-BLD (see Section 3.1 about round-tripping), but ultimately they are results of compromises. While they do simplify translation for some languages, they also make round-trippable translation harder for others. Nevertheless, round-tripping is helped greatly by another interesting feature of RIF: *meta-information*. In RIF-FLD (and in RIF dialects), meta-information can be attached to various syntactic objects at a very fine-grained level. For instance, it can be attached to variables, constants, etc. If enough meta-information is supplied with the RIF document obtained by translation from the language of some other system, then translation from that document back to the original system can be done unambiguously.

RIF-BLD can be easily defined as a specialization of the syntax and semantics of RIF-FLD. The restriction about the uniqueness of context for every symbol can be achieved by requiring that the signatures that are associated with the symbols that are used in RIF-BLD terms can have at most one arrow expression. Other syntactic restrictions are expressed by disallowing negation in rule implications and disjunction in rule heads. The corresponding semantic restrictions largely follow from the restrictions on the syntax. The exact details can be found in [1, Section 6].

6 Conclusions

This paper is an introduction to RIF Framework for Logic Dialects, an extensibility framework that ensures that the current and future dialects of the Rule Interchange Format share common syntactic, semantic, and XML markup apparatus. RIF-FLD is still work in progress: some details may change and additions to the framework should be expected.

Apart from RIF-BLD and the dialect under development for production rule systems, other dialects are being planned. These include the logic programming dialects that support well-founded and stable-model negation, a dialect that supports higher-order extensions as in HiLog [5], and a dialect that extends RIF-BLD with full F-logic [17] support (BLD accommodates only a very small part of F-logic).

The development of the RIF standard is an open process and feedback from experts and users is welcome. All the documents of the working group, meeting agendas, and working lists are publicly available at the group's Web site `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`. The working version of the RIF framework document can be found at the following address: http://www.w3.org/2005/rules/wiki/FLD.

References

1. Boley, H., Kifer, M.: RIF basic logic dialect (October 2007), http://www.w3.org/TR/rif-bld/
2. Boley, H., Kifer, M.: RIF Basic logic dialect. W3C Working Draft (July 2008), http://www.w3.org/TR/rif-fld/
3. Boley, H., Kifer, M.: RIF Framework for logic dialects. W3C Working Draft (July 2008), http://www.w3.org/TR/rif-fld/
4. Bry, F., Eckert, M., Patranjan, P.-L.: Reactivity on the web: Paradigms and applications of the language xchange. Journal of Web Engineering 5(1), 3–24 (2006)
5. Chen, W., Kifer, M., Warren, D.S.: HiLog: A foundation for higher-order logic programming. Journal of Logic Programming 15(3), 187–230 (1993)
6. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 292–322. Plenum Press (1978)
7. Clocksin, W.F., Mellish, C.S.: Programming in Prolog. Springer, Heidelberg (1981)
8. de Sainte Marie, C., Paschke, A.: RIF Production rule dialect. W3C Working Draft (July 2008), http://www.w3.org/TR/rif-prd/
9. Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Owl web ontology language 1.0 reference. Technical report, WWW Consortium (November 2002)
10. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press, London (2001)
11. Fallside, D.C., Walmsley, P.: XML Schema Part 0: Primer 2 edn. Technical report, WWW Consortium (October 2004), http://www.w3.org/TR/xmlschema-0/
12. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Logic Programming: Proceedings of the Fifth Conference and Symposium, pp. 1070–1080 (1988)
13. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3/4), 365–386 (1991)
14. Drools, http://labs.jboss.com/drools/
15. Jess, the rule language for the java platform, http://herzberg.ca.sandia.gov/jess/
16. Kifer, M.: FLORA-2: An object-oriented knowledge base language. The FLORA-2, http://flora.sourceforge.net
17. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of ACM 42, 741–843 (1995)
18. Kozlenkov, A.: PROVA: A Language for Rule-based Java Scripting, Data and Computation Integration, and Agent Programming (May 2005)
19. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987)
20. Polleres, A., Boley, H., Kifer, M.: RIF Datatypes and built-ins. W3C Working Draft (July 2008), http://www.w3.org/TR/rif-dtb/
21. Shoham, Y.: Nonmonotonic logics: meaning and utility. In: Proc. 10th International Joint Conference on Artificial Intelligence, pp. 388–393. Morgan Kaufmann, San Francisco (1987)
22. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of ACM 38(3), 620–650 (1991)

Semantics and Reasoning Algorithms for a Faithful Integration of Description Logics and Rules

Boris Motik

University of Oxford, UK

Abstract. Description logics (DLs) and rule-based systems are two prominent families of knowledge representation formalisms. While DLs are focused on describing and reasoning about conceptual knowledge, rules are focused on answering queries about facts in the knowledge base. So far, research on DLs has been largely isolated from the research on rules. With the advent of the Semantic Web, however, it became apparent that neither formalism alone can cover all the practical use cases. The integration between DLs and rules, however, is technically challenging due to significant differences in the underlying semantic assumptions. In particular, DLs are based on standard first-order semantics and open-world assumption, whereas rules typically employ closed-world semantics based on a variant of circumscription.

In my talk, I shall present an overview of the benefits of integrating DLs and rules in a coherent semantic framework, and shall discuss the main challenges in achieving a tight integration. I shall present an overview of the approaches currently discussed in literature. Finally, I shall present in more detail the approach that is based on the nonmonotonic logic MKNF by Lifschitz. This approach is tight in the sense that DLs and rules are given a common model-theoretic semantics. Furthermore, it is faithful in the sense that it is compatible with the original semantics of both DLs and rules. Finally, it is expressive and can naturally capture many of the existing proposals. I shall discuss the definitions of the semantics of the hybrid formalism, present a decision procedure for a particular decidable fragment, and discuss the complexity bounds of reasoning.

On Combining Description Logic Ontologies and Nonrecursive Datalog Rules

Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma, Italy
rosati@dis.uniroma1.it

Abstract. Reasoning in systems integrating Description Logics (DL) ontologies and Datalog rules is a very hard task, and previous studies have shown undecidability of reasoning in systems integrating (even very simple) DL ontologies with recursive Datalog. However, the results obtained so far constitute a very partial picture of the computational properties of systems combining DL ontologies and Datalog rules. The aim of this paper is to contribute to complete this picture, extending the computational analysis of reasoning in systems integrating ontologies and Datalog rules. More precisely, we first provide a set of decidability and complexity results for reasoning in systems combining ontologies specified in DLs and rules specified in *nonrecursive* Datalog (and its extensions with inequality and negation): such results identify, from the viewpoint of the expressive abilities of the two formalisms, minimal combinations of Description Logics and Datalog in which reasoning is undecidable. Then, we present new results on the decidability and complexity of the so-called *restricted* (or *safe*) integration of DL ontologies and Datalog rules. Our results show that: (1) the unrestricted interaction between DLs and Datalog is computationally very hard even in the absence of recursion in rules; (2) surprisingly, the various "safeness" restrictions, which have been defined to regain decidability of reasoning in the interaction between DLs and recursive Datalog, appear as necessary restrictions even when rules are not recursive.

1 Introduction

Background. The problem of adding rules to ontologies is currently a hot research topic, due to the interest of Semantic Web applications towards the integration of rule-based systems with ontologies. Most of the approaches in this field concern the study of Description Logic (DL) knowledge bases [3] augmented with rules expressed in Datalog and its nonmonotonic extensions [9].

DLs are currently the most used formalisms for building ontologies, and have been proposed as standard languages for the specification of ontologies in the Semantic Web [26]. DLs are a family of knowledge representation formalisms based on first-order logic (FOL). In fact, almost all DLs coincide with decidable fragments of function-free first-order logic with equality, and the language of a DL can be seen as a restricted FOL language over unary and binary predicates and with a controlled form of quantification (actually, DLs are equipped with a special, variable-free syntax). Notably, DLs have been designed to optimize the trade-off between expressive abilities and complexity of reasoning, hence the computational properties of DLs have been extensively studied [3].

From the knowledge representation viewpoint, Datalog is somehow "complementary" to DLs. Indeed, with respect to DLs, Datalog allows for using predicates of arbitrary arity, the explicit use of variables, and the ability of expressing more powerful queries. Moreover, its nonmonotonic features (in particular, the negation-as-failure operator not) allow for expressing default rules and forms of closed-world reasoning.

Problem Studied. Unfortunately, reasoning in systems integrating DLs and Datalog is a very hard task, and well-known previous results have shown undecidability of reasoning in systems fully integrating (even very simple) DL ontologies with Datalog rules. In fact, in general this combination does not preserve decidability, i.e., starting from a DL knowledge base in which reasoning is decidable and a set of rules in which reasoning is decidable, reasoning in the knowledge base obtained by integrating these two components may not be a decidable problem.

To avoid undecidability of reasoning, practically all decidable approaches to integrating ontologies and rules impose (either at the syntactic or at the semantic level) specific conditions which restrict the interaction between the rules and the ontology. Such restrictions were mainly introduced to keep reasoning decidable in the presence of recursion in Datalog rules.

However, the results obtained so far [20,11,18,23,27,28,10] actually constitute a very partial picture of the computational properties of systems combining DL ontologies and Datalog rules. In particular, the computational properties of systems combining DL ontologies and the class of *nonrecursive* Datalog rules are mostly unknown. The only known studies related to this topic are the work on CARIN [20], which has shown decidability of nonrecursive positive Datalog with the DL $\mathcal{ALCNR}$, and the studies on conjunctive query answering in DLs (see e.g. [7,24,25,14,15]), which are indirectly related to integrating Datalog and DLs (since conjunctive queries can be seen as nonrecursive Datalog programs consisting of a single rule).

Contribution. The aim of this paper is to contribute to fill this gap, extending the computational analysis of reasoning in systems integrating ontologies and Datalog rules. More precisely, our contributions can be summarized as follows:

– We first provide a set of decidability and complexity results for reasoning in systems combining ontologies specified in (different classes of) DLs and rules specified in (different classes of) *nonrecursive* Datalog (and its extensions with inequality or negation). Such results identify, from the viewpoint of the expressive abilities of the two formalisms, minimal combinations of Description Logics and (nonmonotonic) Datalog in which reasoning is undecidable. A summary of the results obtained is reported in Figure 2 (Section 4).
– Then, we present new results on the decidability and complexity of the restricted integration of DL ontologies and Datalog rules. In particular, we consider the so-called "weakly DL-safe" interaction between rules and DL ontologies [28], which is currently one of the most expressive decidable combinations of DLs and rules: we extend the framework of [28] to deal with both negation of DL predicates and

the presence of inequality, and provide new decidability and complexity results for such a class of weakly DL-safe Datalog rules.

Besides constituting one of the first refined computational analyses taking into account the expressive power of both the DL language and the rule language (the only similar study which we are aware of is [20]), the above results imply the following consequences:

- the unrestricted interaction of DLs and Datalog is computationally very hard even in the absence of recursion in rules. This contrasts with the general opinion (suggested by the results in [20]) that the presence of recursion in rules is necessary in order to rise the undecidability issue in their combination with DL ontologies;
- surprisingly, the "safeness" restrictions, which have been defined to regain decidability in the interaction between DLs and recursive Datalog, appear as necessary restrictions even when rules are not recursive.

Structure of the Paper. In Section 2, we briefly recall the basics of Description Logics and Datalog. In Section 3, we formally define syntax and semantics of systems integrating DLs and Datalog. In Section 4, we consider the full integration of DLs and rules, and present a set of undecidability and hardness results for reasoning in systems fully combining DLs and Datalog rules. In Section 5, we focus on weakly DL-safe systems, which are based on a restricted form of interaction between DLs and rules, extend them to the presence of inequality atoms, and present a computational analysis of reasoning in such systems. Finally, we conclude in Section 6. Due to space limits, in the present version of the paper we provide proof sketches of the theorems.

2 Description Logics and Datalog

In this section we briefly introduce Description Logics and Datalog.

Description Logics. We now briefly recall the basics of Description Logics (DLs) and introduce the following DLs: (i) three prominent *tractable* DLs, i.e., *DL-Lite*$_{RDFS}$, *DL-Lite*$_{R}$ and $\mathcal{EL}$; (ii) the "classical" and *moderately expressive* DL $\mathcal{ALC}$; (iii) two *very expressive* DLs, i.e., $\mathcal{SHIQ}$ and $\mathcal{DLR}$. We refer to [3] for a more detailed introduction to DLs.

We start from an alphabet of concept names, an alphabet of role names and an alphabet of constant names. Concepts correspond to unary predicates in FOL, roles correspond to binary predicates, and constants corresponds to FOL constants.

Starting from concept and role names, *concept expressions* and *role expressions* can be constructed, based on a formal syntax. Different DLs are based on different languages concept and role expressions. Details on the concept and role languages for the DLs considered in this paper are reported below.

A *concept inclusion* is an expression of the form $C_1 \sqsubseteq C_2$, where C_1 and C_2 are concept expressions. Similarly, a *role inclusion* is an expression of the form $R_1 \sqsubseteq R_2$, where R_1 and R_2 are role expressions.

DL	concept expressions	role expressions	TBox axioms
DL-Lite$_{RDFS}$	$C_L ::= A \mid \exists R$ $C_R ::= A$	$R ::= P \mid P^-$	$C_L \sqsubseteq C_R$ $R_1 \sqsubseteq R_2$
DL-Lite$_R$	$C_L ::= A \mid \exists R$ $C_R ::= A \mid \neg C_R \mid \exists R$	$R ::= P \mid P^-$	$C_L \sqsubseteq C_R$ $R_1 \sqsubseteq R_2$
$\mathcal{EL}$	$C ::= A \mid C_1 \sqcap C_2 \mid \exists P.C$	$R ::= P$	$C_1 \sqsubseteq C_2$
$\mathcal{ALC}$	$C ::= A \mid C_1 \sqcap C_2 \mid \neg C \mid \exists P.C$	$R ::= P$	$C_1 \sqsubseteq C_2$
$\mathcal{SHIQ}$	$C ::= A \mid \neg C \mid C_1 \sqcap C_2 \mid (\geq n\, R\, C)$	$R ::= P \mid P^-$	$C_1 \sqsubseteq C_2$ $R_1 \sqsubseteq R_2$ Trans(R)

Fig. 1. Abstract syntax of the DLs studied in the paper

An *instance assertion* is an expression of the form $A(a)$ or $P(a, b)$, where A is a concept name, P is a role name, and a, b are constant names. We do not consider complex concept and role expressions in instance assertions, since in this paper we are interested in data complexity of reasoning (see Section 4).

A *DL knowledge base (KB)* is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, called the *TBox*, is a set of concept and role inclusions, and $\mathcal{A}$, called the *ABox*, is a set of instance assertions.

The DLs mainly considered in this paper are the following:

– *DL-Lite*$_{RDFS}$, which corresponds to the "DL fragment" of RDFS [1], the schema language for RDF (see also [16]);
– *DL-Lite*$_R$ [5], a tractable DL which is tailored for efficient reasoning and query answering in the presence of very large ABoxes;
– $\mathcal{EL}$ [2], a prominent tractable DL;
– $\mathcal{ALC}$, a very well-known DL which corresponds to multimodal logic K_n [3];
– $\mathcal{SHIQ}$, a very expressive DL which constitutes the basis of the OWL family of DLs adopted as standard languages for ontology specification in the Semantic Web [26].

The syntax of the above DLs is summarized in Figure 1, in which the symbol A denotes a concept name and the symbol P denotes a role name (in addition to concept and role inclusions, $\mathcal{SHIQ}$ also allows for TBox axioms of the form Trans(R), which state transitivity of the role R).

We will also mention the DL $\mathcal{DLR}$ [7], which informally extends $\mathcal{SHIQ}$ (without transitive roles) through the use of n-ary relations, and for which decidability results on query answering are known (we refer to [7] for details on the syntax of $\mathcal{DLR}$, which is quite different from the other DLs due to the usage of relations of arbitrary arity).

The above mentioned DLs verify the following ordering with respect to their relative expressive power (see [3] for details): *DL-Lite*$_{RDFS}$ $\subset$ *DL-Lite*$_R$ $\subset \mathcal{SHIQ} \subset \mathcal{DLR}$; and $\mathcal{EL} \subset \mathcal{ALC} \subset \mathcal{SHIQ}$.

We give the semantics of DLs through the well-known translation ρ_{fol} of DL knowledge bases into FOL theories with counting quantifiers (see [3]).

$$\begin{aligned}
\rho_{fol}(\langle \mathcal{T}, \mathcal{A} \rangle) &= \rho_{fol}(\mathcal{T}) \cup \rho_{fol}(\mathcal{A}) \\
\rho_{fol}(C_1 \sqsubseteq C_2) &= \forall x. \rho_{fol}(C_1, x) \rightarrow \rho_{fol}(C_2, x) \\
\rho_{fol}(R_1 \sqsubseteq R_2) &= \forall x, y. \rho_{fol}(R_1, x, y) \rightarrow \rho_{fol}(R_2, x, y) \\
\rho_{fol}(\mathrm{Trans}(R)) &= \forall x, y, z. \rho_{fol}(R, x, y) \wedge \rho_{fol}(R, y, z) \rightarrow \rho_{fol}(R, x, z) \\
\rho_{fol}(A, x) &= A(x) \\
\rho_{fol}(\neg C, x) &= \neg \rho_{fol}(C, x) \\
\rho_{fol}(C_1 \sqcap C_2, x) &= \rho_{fol}(C_1, x) \wedge \rho_{fol}(C_2, x) \\
\rho_{fol}(\exists R, x) &= \exists y. \rho_{fol}(R, x, y) \\
\rho_{fol}(\exists R.C, x) &= \exists y. \rho_{fol}(R, x, y) \wedge \rho_{fol}(C, y) \\
\rho_{fol}((\geq n\, R\, C), x) &= \exists^{\geq n} y. \rho_{fol}(R, x, y) \wedge \rho_{fol}(C, y) \\
\rho_{fol}(P, x, y) &= P(x, y) \\
\rho_{fol}(P^-, x, y) &= P(y, x)
\end{aligned}$$

An interpretation of $\mathcal{K}$ is a classical FOL interpretation for $\rho_{fol}(\mathcal{K})$, where constants and predicates are interpreted over a non-empty interpretation domain which may be finite or countably infinite. Actually, in this paper we adopt the standard names assumption, i.e.: (i) we assume a countably infinite set of constant symbols Γ; (ii) the interpretation domain Δ is countably infinite and is the same for every interpretation; (iii) the interpretation of constants in Γ is the same in every interpretation and is given by a one-to-one correspondence between Γ and Δ. Such an assumption is necessary for the nonmonotonic semantics defined in Section 3: however, we point out that all the results presented in this paper under the first-order semantics (i.e., the results for FOL-satisfiability) also hold in the absence of the standard names assumption.

A *model* of a DL KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a FOL model of $\rho_{fol}(\mathcal{K})$. We say that $\mathcal{K}$ is *satisfiable* if $\mathcal{K}$ has a model.

Disjunctive Datalog. In this section be briefy recall disjunctive Datalog [9], denoted by Datalog$^{\neg\vee}$, which is the well-known nonmonotonic extension of Datalog with negation as failure and disjunction.

We start from a predicate alphabet, a constant alphabet, and a variable alphabet. An *atom* is an expression of the form $p(X)$, where p is a predicate of arity n and X is a n-tuple of variables and constants. If no variable symbol occurs in X, then $p(X)$ is called a *ground atom* (or *fact*). A Datalog$^{\neg\vee}$ rule R is an expression of the form

$$\alpha_1 \vee \ldots \vee \alpha_n \;\leftarrow\; \beta_1, \ldots, \beta_m, \mathit{not}\, \gamma_1, \ldots, \mathit{not}\, \gamma_k, t_1 \neq t'_1, \ldots, t_h \neq t'_h \qquad (1)$$

where each α_i, β_i, γ_i is an atom, each t_i, t'_i, is either a variable or a constant, and every variable occurring in R must appear in at least one of the atoms $\beta_1, \ldots, \beta_m$. This last condition is known as the *Datalog safeness* condition for variables. The variables occurring in the atoms $\alpha_1, \ldots, \alpha_n$ are called the *head variables* of R. If $n = 0$, we call R a *constraint*.

A *Datalog*$^{\neg\vee}$ program is a set of Datalog$^{\neg\vee}$ rules. If, for all $R \in \mathcal{P}$, $k = 0$ and $h = 0$, $\mathcal{P}$ is called a *positive disjunctive Datalog* program. If, for all $R \in \mathcal{P}$, $n \leq 1$, $k = 0$ and $h = 0$, $\mathcal{P}$ is called a *positive Datalog* program. If there are no occurrences of variable symbols in a rule R, then R is called a *ground* rule. A *ground* program is a program containing only ground rules.

The *dependency graph* of a program P is a graph whose nodes are the predicates of P and in which there is an edge from p_1 to p_2 if there is a rule r in P such that p_2 occurs in the body of r and p_1 occurs in the head of r. A program P is *recursive* if its dependency graph contains a cycle. Otherwise, P is called *nonrecursive*.

The semantics of disjunctive Datalog is given in terms of *stable models* of a program $\mathcal{P}$. Due to space limitations, we refer to [9] for details on such semantics: however, in the following we will provide a detailed definition of such semantics in the more general framework of r-hybrid KBs integrating DLs and disjunctive Datalog.

3 R-Hybrid KBs

In this section we present the framework of r-hybrid KBs which integrate DLs with disjunctive Datalog. More precisely, we slightly extend the framework of r-hybrid knowledge bases presented in [27] to the presence of both inequality atoms and negation of DL predicates in rules.

Syntax. From the syntactic viewpoint, integrating a DL with (disjunctive) Datalog simply means the possibility of writing a *hybrid* knowledge base $\mathcal{H}$ containing a DL KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a disjunctive Datalog program $\mathcal{P}$ (i.e., $\mathcal{H} = (\mathcal{K}, \mathcal{P})$) where $\mathcal{K}$ and $\mathcal{P}$ share both the alphabet of predicates and the alphabet of constants. However, for technical reasons related to the subsequent definition of the nonmonotonic semantics, we distinguish the predicates occurring only in $\mathcal{P}$, which we call *Datalog predicates*, from the ones occurring both in $\mathcal{K}$ and in $\mathcal{P}$, or even only in $\mathcal{K}$, which we call *DL predicates*. In the following, we denote by $\Sigma_C \cup \Sigma_R$ the set of DL predicates, and denote by Σ_D the set of Datalog predicates. Formally, a rule R in $\mathcal{P}$ is a rule of the form (1) over both DL-predicates and Datalog predicates. An atom whose predicate is a DL predicate is called a *DL atom*, while an atom whose predicate is a Datalog predicate is called a *Datalog atom*.

First-Order Semantics. According to a semantic approach based on classical logic, the hybrid knowledge base can be considered as a first-order theory, by interpreting Datalog rules as first-order implications. More specifically, let R be the Datalog$^{\neg\vee}$ rule of the form (1). Then, we denote by $FO(R)$ the first-order sentence

$$\forall x_1, \ldots, x_p. \, \beta_1 \wedge \ldots \wedge \beta_m \wedge \neg\gamma_1 \wedge \ldots \wedge \neg\gamma_k \wedge t_1 \neq t_1' \wedge \ldots \wedge t_h \neq t_h' \rightarrow \alpha_1 \vee \ldots \vee \alpha_n$$

where $x_1, \ldots, x_p$ are all the variable symbols appearing in R. Given a Datalog$^{\neg\vee}$ program $\mathcal{P}$, we denote by $FO(\mathcal{P})$ the set of first-order sentences $\{FO(R) \mid R \in \mathcal{P}\}$.

Finally, the semantics of a knowledge base $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ composed of a DL-KB $\mathcal{K}$ and a Datalog program $\mathcal{P}$ is given by the first-order theory $FO(\mathcal{H})$ corresponding to the union of $FO(\mathcal{P})$ and the first-order translation $FO(\mathcal{K})$ of $\mathcal{K}$: in particular, we say that $\mathcal{H}$ is *FOL-satisfiable* if $FO(\mathcal{H})$ has a model (which is called *FOL-model* of $\mathcal{H}$), and we say that a ground atom g is FOL-entailed by $\mathcal{H}$, denoted by $\mathcal{H} \models_{FOL} g$ iff, for each FOL-model $\mathcal{I}$ of $\mathcal{H}$, $\mathcal{I}$ satisfies g.

Nonmonotonic Semantics. We now recall the nonmonotonic semantics for r-hybrid KBs presented in [27], which is a "conservative extension" of both the open-world semantics (classical FOL models) of DLs and the closed-world semantics (stable models) of disjunctive Datalog.

Given an interpretation $\mathcal{I}$ and a predicate alphabet Σ, we denote by $\mathcal{I}_\Sigma$ the projection of $\mathcal{I}$ to Σ, i.e., $\mathcal{I}_\Sigma$ is obtained from $\mathcal{I}$ by restricting it to the interpretation of the predicates in Σ.

The *ground instantiation of* $\mathcal{P}$, denoted by $gr(\mathcal{P})$, is the program obtained from $\mathcal{P}$ by replacing every rule R in $\mathcal{P}$ with the set of rules obtained by applying all possible substitutions of variables in R with constants in Γ.

Given an interpretation $\mathcal{I}$ of an alphabet of predicates $\Sigma' \subset \Sigma$, and a ground program $\mathcal{P}_g$ over the predicates in Σ, the *projection of* $\mathcal{P}_g$ *with respect to* $\mathcal{I}$, denoted by $\Pi(\mathcal{P}_g, \mathcal{I})$, is the ground program obtained from $\mathcal{P}_g$ as follows. For each rule $R \in \mathcal{P}_g$:

- delete R if there exists an atom $r(t)$ in the head of R such that $r \in \Sigma'$ and $t^\mathcal{I} \in r^\mathcal{I}$;
- delete each atom $r(t)$ in the head of R such that $r \in \Sigma'$ and $t^\mathcal{I} \notin r^\mathcal{I}$;
- delete R if: either (i) there exists an atom $r(t)$ in the body of R such that $r \in \Sigma'$ and $t^\mathcal{I} \notin r^\mathcal{I}$; or (ii) there exists a negated atom $not\, r(t)$ in the body of R such that $r \in \Sigma'$ and $t^\mathcal{I} \in r^\mathcal{I}$;
- delete each atom $r(t)$ in the body of R such that $r \in \Sigma'$ and $t^\mathcal{I} \in r^\mathcal{I}$;
- delete each negated atom $not\, r(t)$ in the body of R such that $r \in \Sigma'$ and $t^\mathcal{I} \notin r^\mathcal{I}$.

Informally, the projection of $\mathcal{P}_g$ with respect to $\mathcal{I}$ corresponds to evaluating $\mathcal{P}_g$ with respect to $\mathcal{I}$, thus eliminating from $\mathcal{P}_g$ every atom whose predicate is interpreted in $\mathcal{I}$. Thus, when $\Sigma' = \Sigma_C \cup \Sigma_R$, all occurrences of DL predicates are eliminated in the projection of $\mathcal{P}_g$ with respect to $\mathcal{I}$, according to the evaluation in $\mathcal{I}$ of the atoms with DL predicates occurring in $\mathcal{P}_g$.

Given two interpretations $\mathcal{I}_1, \mathcal{I}_2$ of the set of predicates Σ, we write $\mathcal{I}_1 \subset_\Sigma \mathcal{I}_2$ if (i) for each $p \in \Sigma$ and for each tuple t of constants from Γ, if $t^{\mathcal{I}_1} \in p^{\mathcal{I}_1}$ then $t^{\mathcal{I}_2} \in p^{\mathcal{I}_2}$, and (ii) there exist $p \in \Sigma$ and tuple t of constants from Γ such that $t^{\mathcal{I}_1} \notin p^{\mathcal{I}_1}$ and $t^{\mathcal{I}_2} \in p^{\mathcal{I}_2}$.

Given a positive disjunctive ground Datalog$^{\neg\vee}$ program $\mathcal{P}$ over an alphabet of predicates Σ and an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ is a *minimal model* of $\mathcal{P}$ if: (i) $\mathcal{I}$ satisfies the first-order translation $FO(\mathcal{P})$ of $\mathcal{P}$; (ii) there is no interpretation $\mathcal{I}'$ such that $\mathcal{I}'$ satisfies $FO(\mathcal{P})$ and $\mathcal{I}' \subset_\Sigma \mathcal{I}$.

Given a ground Datalog$^{\neg\vee}$ program $\mathcal{P}$ and an interpretation $\mathcal{I}$ for $\mathcal{P}$, the *GL-reduct* [12] of $\mathcal{P}$ with respect to $\mathcal{I}$, denoted by $GL(\mathcal{P}, \mathcal{I})$, is the positive disjunctive ground program obtained from $\mathcal{P}$ as follows. For each rule $R \in \mathcal{P}$:

1. delete R if either there exists a negated atom $not\, r(t)$ in the body of R such that $t^\mathcal{I} \in r^\mathcal{I}$, or there exists an inequality $c \neq c$ in the body of R;
2. delete each negated atom $not\, r(t)$ in the body of R such that $t^\mathcal{I} \notin r^\mathcal{I}$ and delete each inequality $c \neq d$ where c and d are distinct constants.

Given a ground Datalog$^{\neg\vee}$ program $\mathcal{P}$ and an interpretation $\mathcal{I}$, $\mathcal{I}$ is a *stable model* for $\mathcal{P}$ iff $\mathcal{I}$ is a minimal model of $GL(\mathcal{P}, \mathcal{I})$.

Definition 1. *An interpretation $\mathcal{I}$ of $\Sigma_C \cup \Sigma_R \cup \Sigma_D$ is a NM-model for $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ if the following conditions hold: (i) $\mathcal{I}_{\Sigma_C \cup \Sigma_R}$ satisfies $\mathcal{K}$; (ii) $\mathcal{I}_{\Sigma_D}$ is a stable model for $\Pi(gr(\mathcal{P}), \mathcal{I}_{\Sigma_C \cup \Sigma_R})$. $\mathcal{H}$ is called* NM-satisfiable *if $\mathcal{H}$ has at least one NM-model.*

We say that a ground atom g is NM-entailed by $\mathcal{H}$, denoted by $\mathcal{H} \models_{NM} g$ iff, for each NM-model $\mathcal{I}$ of $\mathcal{H}$, $\mathcal{I}$ satisfies g.

According to the above semantics, DL predicates are interpreted under the open-world assumption, while Datalog predicates are interpreted under the closed-world assumption of disjunctive Datalog and Answer Set Programming. As a consequence, negation of DL predicates in rule bodies is interpeted as classical (monotonic) negation, while negation of Datalog predicates is interpreted as nonmonotonic negation (negation as failure under stable model semantics).

Reasoning: General Properties. Notice that, under the above NM semantics (as well as under the FOL semantics), entailment can be reduced to unsatisfiability, since it is possible to express constraints (i.e., rules with empty head) in the Datalog program. More precisely, the following property holds.

Proposition 1. *Let $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ be a r-hybrid KB and let g be a ground atom. Then, $\mathcal{H} \models_{NM} g$ (respectively, $\mathcal{H} \models_{FOL} g$) iff the r-hybrid KB $(\mathcal{K}, \mathcal{P} \cup \{\leftarrow g\})$ is NM-unsatisfiable (respectively, FOL-unsatisfiable).*

Then, we show that, when there are no negated Datalog atoms in the bodies of rules, the above two semantics are equivalent with respect to the satisfiability problem. The following property extends an analogous one shown in [28].

Proposition 2. *Let $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ be a r-hybrid KB, where $\mathcal{P}$ is such that there are no occurrences of negated Datalog atoms in $\mathcal{P}$. Then, $\mathcal{H}$ is FOL-satisfiable iff $\mathcal{H}$ is NM-satisfiable.*

4 Results for Nonrecursive Rules

In this section we present a set of new results on the decidability and complexity of reasoning in r-hybrid KBs, under both FOL-semantics and NM-semantics.

We have conducted our computational analysis on the following subclasses of *non-recursive* and *nondisjunctive* Datalog programs:

- *NR-Datalog* = nonrecursive positive Datalog, i.e., nonrecursive rules of the form (1) where $n \leq 1$, $k = 0$, $h = 0$;
- *NR-Datalog*$^{\neq}$ = nonrecursive positive Datalog with inequality, i.e., nonrecursive rules of the form (1) where $n \leq 1$, $k = 0$;
- *NR-Datalog*$_s^{\neq}$ = single-rule nonrecursive positive Datalog with inequality (i.e., *NR-Datalog*$^{\neq}$ programs consisting of a single rule);
- *NR-Datalog*$^{\neg}$ = nonrecursive Datalog with negation, i.e., nonrecursive rules of the form (1) where $n \leq 1$, $h = 0$;
- *NR-Datalog*$_s^{\neg}$ = single-rule nonrecursive Datalog with negation (i.e., *NR-Datalog*$^{\neg}$ programs consisting of a single rule);
- *NR-Datalog*$^{\neg A}$ = nonrecursive Datalog with "atomic" negation, i.e., *NR-Datalog*$^{\neg}$ programs such that predicates occurring in negated atoms cannot occur in rule heads.

Moreover, throughout this section we impose the further restriction that programs are such that *DL predicates do not occur in the head of rules*. We call *head-DL-free* the programs satisfying the above restriction. Such a restriction strengthens the lower bounds and undecidability results which are presented below.

Furthermore, we remark that we focus on *data complexity* of satisfiability, which in the framework of r-hybrid KBs ($\mathcal{H} = (\mathcal{K}, \mathcal{P})$ with $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$) corresponds to the analysis of the computational complexity of the problem when we only consider the size of the ABox $\mathcal{A}$ and of the EDB of $\mathcal{P}$, i.e., the set of facts contained in $\mathcal{P}$.

Finally, we point out that most of the proofs of the following theorems are obtained by exploiting and extending the proofs of recent results on query answering in DLs, in particular the results in [6,29].

We start by analyzing r-hybrid KBs with *NR-Datalog* programs.

Theorem 1. *Let $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ be a r-hybrid KB such that $\mathcal{P}$ is a head-DL-free NR-Datalog program. Then, under both FOL semantics and NM semantics:*

- *when $\mathcal{K}$ is either a DL-Lite$_{RDFS}$ KB or a DL-Lite$_R$ KB, deciding satisfiability of $\mathcal{H}$ is in* LOGSPACE *with respect to data complexity;*
- *when $\mathcal{K}$ is an $\mathcal{EL}$ KB, deciding satisfiability of $\mathcal{H}$ is* PTIME*-complete with respect to data complexity.*

Proof (sketch). First, observe that by Proposition 2 FOL-satisfiability and NM-satisfiability coincide for the class of r-hybrid KBs considered. Then, for *DL-Lite*$_{RDFS}$ and *DL-Lite*$_R$ the thesis follows from the complexity results on answering unions of conjunctive queries in *DL-Lite*$_R$ [6] and from the fact that it is possible to reduce unsatisfiability of $(\mathcal{K}, \mathcal{P})$, where $\mathcal{P}$ is nonrecursive, to the evaluation of unions of conjunctive queries over $\mathcal{K}$. In the case of $\mathcal{EL}$, the thesis follows from a similar argument and from the computational properties of answering unions of conjunctive queries in $\mathcal{EL}$ [29, Theorem 4].

Then, we provide the following computational characterization of satisfiability in the presence of *NR-Datalog*$_s^{\neq}$ programs.

Theorem 2. *Let $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ be a r-hybrid KB such that $\mathcal{P}$ is a head-DL-free NR-Datalog$_s^{\neq}$ program. Then, under both FOL semantics and NM semantics:*

- *when $\mathcal{K}$ is a DL-Lite$_{RDFS}$ KB, deciding satisfiability of $\mathcal{H}$ is in* LOGSPACE *with respect to data complexity;*
- *when $\mathcal{K}$ is an $\mathcal{EL}$ KB, deciding satisfiability of $\mathcal{H}$ is* PTIME*-complete with respect to data complexity;*
- *when $\mathcal{K}$ is a DL-Lite$_R$ KB, deciding satisfiability of $\mathcal{H}$ is* NP*-hard with respect to data complexity;*
- *when $\mathcal{K}$ is an $\mathcal{ALC}$ KB, satisfiability of $\mathcal{H}$ is undecidable.*

Proof (sketch). First, observe that by Proposition 2 FOL-satisfiability and NM-satisfiability coincide for the class of r-hybrid KBs considered. Then, for *DL-Lite*$_{RDFS}$ and $\mathcal{EL}$ the thesis is a consequence of a property analogous to [29, Theorem 7], and to the data complexity of answering conjunctive queries in those DLs [6,29], while for

DL-Lite$_R$ the proof is by reduction from satisfiability of a 3-CNF propositional formula, in a way analogous to [29, Theorem 6]. Finally, in the case of $\mathcal{ALC}$ the proof is by reduction from the unbounded tiling problem [4], in a way analogous to [29, Theorem 5].

We then analyze reasoning in r-hybrid KBs with *NR-Datalog*$^{\neq}$ programs.

Theorem 3. *Let* $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ *be a r-hybrid KB such that* $\mathcal{P}$ *is a head-DL-free NR-Datalog*$^{\neq}$ *program. Then, under both FOL semantics and NM semantics:*

- *when* $\mathcal{K}$ *is a DL-Lite*$_{RDFS}$ *KB, deciding satisfiability of* $\mathcal{H}$ *is in* LOGSPACE *with respect to data complexity;*
- *when* $\mathcal{K}$ *is either a DL-Lite*$_R$ *KB or an* $\mathcal{EL}$ *KB, satisfiability of* $\mathcal{H}$ *is undecidable.*

Proof (sketch). Again, we start by observing that by Proposition 2 FOL-satisfiability and NM-satisfiability coincide for the class of r-hybrid KBs considered. Then, for *DL-Lite*$_{RDFS}$ the proof is obtained by extending the result in [29, Theorem 11], while in the case of both *DL-Lite*$_R$ and $\mathcal{EL}$ the proof is obtained by reducing the word problem for semigroups to satisfiability in such DLs, in a way analogous to Theorem 8 and Theorem 9 of [29].

Next, we are able to prove the following results for r-hybrid KBs with *NR-Datalog*$_s^{\neg}$ programs.

Theorem 4. *Let* $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ *be a r-hybrid KB such that* $\mathcal{P}$ *is a head-DL-free NR-Datalog*$_s^{\neg}$ *program. Then, under both FOL semantics and NM semantics:*

- *when* $\mathcal{K}$ *is a DL-Lite*$_{RDFS}$ *KB, deciding satisfiability of* $\mathcal{H}$ *is in* LOGSPACE *with respect to data complexity;*
- *when* $\mathcal{K}$ *is an* $\mathcal{EL}$ *KB, deciding satisfiability of* $\mathcal{H}$ *is* PTIME*-complete with respect to data complexity;*
- *when* $\mathcal{K}$ *is a DL-Lite*$_R$ *KB, deciding satisfiability of* $\mathcal{H}$ *is* NP*-hard with respect to data complexity;*
- *when* $\mathcal{K}$ *is an* $\mathcal{ALC}$ *KB, satisfiability of* $\mathcal{H}$ *is undecidable.*

Proof (sketch). First, we consider the case of FOL-satisfiability. For *DL-Lite*$_{RDFS}$ and $\mathcal{EL}$ the proof is obtained from [29, Theorem 14] and from the data complexity of answering conjunctive queries in those DLs [6,29], while for *DL-Lite*$_R$ the proof is by reduction from satisfiability of a 3-CNF propositional formula, in a way analogous to [29, Theorem 13]. Finally, in the case of $\mathcal{ALC}$ the proof is by reduction from the unbounded tiling problem [4], in a way analogous to [29, Theorem 12]. The above results can be easily extended to the case of NM-satisfiability: in particular, the above reductions used for *DL-Lite*$_R$ and $\mathcal{ALC}$ do not employ negated Datalog atoms in rules, hence by Proposition 2 such reductions also prove the thesis under the NM semantics.

Finally, we consider *NR-Datalog*$^{\neg A}$ programs, and provide the following results.

Theorem 5. *Let* $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ *be a r-hybrid KB such that* $\mathcal{P}$ *is a head-DL-free NR-Datalog*$^{\neg A}$ *program. Then, under both FOL semantics and NM semantics:*

	NR-Datalog	*NR-Datalog*$^{\neq}_{s}$	*NR-Datalog*$^{\neq}$	*NR-Datalog*$^{\neg}_{s}$	*NR-Datalog*$^{\neg A}$	*NR-Datalog*$^{\neg}$
DL-Lite$_{RDFS}$	$\leq$LOGSPACE	$\leq$LOGSPACE	$\leq$LOGSPACE	$\leq$LOGSPACE	$=$NP	UNDEC.
DL-Lite$_{R}$	$\leq$LOGSPACE	$\geq$NP	UNDEC.	$\geq$NP	UNDEC.	UNDEC.
$\mathcal{EL}$	$=$PTIME	$=$PTIME	UNDEC.	$=$PTIME	UNDEC.	UNDEC.
from $\mathcal{ALC}$ to $\mathcal{SHIQ}$	$=$NP	UNDEC.	UNDEC.	UNDEC.	UNDEC.	UNDEC.
$\mathcal{DLR}$	DECID., $\geq$NP	UNDEC.	UNDEC.	UNDEC.	UNDEC.	UNDEC.

Fig. 2. Decidability/data complexity of both FOL-satisfiability and NM-satisfiability in r-hybrid KBs (head-DL-free programs)

– *when $\mathcal{K}$ is a DL-Lite$_{RDFS}$ KB, deciding satisfiability of $\mathcal{H}$ is* NP-*hard with respect to data complexity;*
– *when $\mathcal{K}$ is either a DL-Lite$_{R}$ KB or an $\mathcal{EL}$ KB, satisfiability of $\mathcal{H}$ is undecidable.*

Proof (sketch). First, we consider the case of FOL-satisfiability. For *DL-Lite*$_{RDFS}$ the proof is obtained from [29, Theorem 16], while for *DL-Lite*$_{R}$ and $\mathcal{EL}$ the proof is by reduction from the unbounded tiling problem, in a way analogous to [29, Theorem 15]. Finally, the above reductions do not employ negated Datalog atoms in rules, hence by Proposition 2 such reductions also prove the thesis under the NM semantics.

The table displayed in Figure 2 summarizes the results presented in this section. In the table, each column corresponds to a different rule language, while each row corresponds to a different DL. Each cell reports the data complexity of satisfiability (both under FOL semantics and under NM semantics) in the corresponding combination of DL and rule language. If the problem is decidable, then hardness ($\geq$) and/or membership ($\leq$) and/or completeness ($=$) results are reported.

More precisely, observe that:

– the results for *NR-Datalog* programs follow from Theorem 1 and from the results in [6,13];
– the well-known translation of arbitrary first-order queries in *NR-Datalog*$^{\neg}$ allows for reducing satisfiability of first-order sentences to satisfiability of r-hybrid KBs with *NR-Datalog*$^{\neg}$ programs for *any* choice of the DL language, which immediately implies undecidability of reasoning in this class of r-hybrid KBs.

Finally, due to the correspondence between unsatisfiability and entailment in r-hybrid KBs illustrated in Section 3 (Proposition 1), it is also immediate to turn these results (obtained for satisfiability of programs with constraints) into results for skeptical entailment (also for classes of programs without constraints).

5 Results for Weakly DL-Safe Rules

In this section we consider the weakly DL-safe integration of DLs and disjunctive Datalog. More precisely, we extend the weak DL-safeness restriction defined in the framework of $\mathcal{DL}$+*log* [28] to the r-hybrid KBs defined in Section 3, thus extending the setting presented in [28] by considering the presence of inequality and of negation of

DL predicates. Then, we extend the computational results presented in [28] to such a class of r-hybrid KBs.

Weak DL-safeness is formally defined as follows.

Definition 2. *Given a r-hybrid KB $\mathcal{H} = (\mathcal{K}, \mathcal{P})$, we say that $\mathcal{P}$ is* weaky DL-safe *if every rule R in $\mathcal{P}$ of the form (1) is such that, for every variable x appearing in R, either x occurs in a positive Datalog atom in the body of R, or x only occurs in positive DL atoms in the body of R.*

In other words, weak DL-safeness imposes (besides the usual Datalog safeness) the following condition: every variable that is either a head variable or a variable occurring in a negated atom or in an inequality occurs in a positive Datalog atom in the body of the rule. Such a restriction only constrains the interaction between the DL KB and the Datalog program, in the sense that neither it imposes any additional restriction on the rules if the DL KB is empty, nor it imposes any restriction on the DL KB.

We now show decidability of reasoning in r-hybrid KBs under the above restriction. To this aim, we start from the algorithm for NM-satisfiability in $\mathcal{DL}$+*log* presented in [28] and extend it to the broader class of rules considered here. Due to space limits, we do not report details on the algorithm, which is actually very similar to the one reported in [28]. Such an algorithm checks satisfiability of a r-hybrid KB by solving a finite number of *Boolean CQ/UCQ containment* problems in DLs. Boolean CQ/UCQ containment is the problem of checking the containment between two queries q_1 and q_2 with respect to a DL KB $\mathcal{K}$, where q_1 is a Boolean conjunctive query and q_2 is a Boolean union of conjunctive queries (this problem is also known as *existential entailment* [20]).

Based on such an algorithm, we are able to extend the general decidability result of [28] to the present class of r-hybrid KBs. Formally:

Theorem 6. *Let $\mathcal{DL}$ be a description logic and let $\mathcal{H} = (\mathcal{K}, \mathcal{P})$ be a r-hybrid KB, where $\mathcal{K}$ is a $\mathcal{DL}$ KB and $\mathcal{P}$ is a weakly DL-safe Datalog$^{\neg\vee}$ program. NM-satisfiability (as well as FOL-satisfiability) of $\mathcal{H}$ is decidable iff Boolean CQ/UCQ containment is decidable in $\mathcal{DL}$.*

In particular, the above theorem and the results on CQ/UCQ containment in DLs presented in [20,7,15,25] imply the following property: *for all the DLs studied in this paper, NM-satisfiability (as well as FOL-satisfiability) of weakly DL-safe r-hybrid KBs is decidable.*

Moreover, based on the above cited results and on our technique for NM-satisfiability, we are able to provide a computational characterization of r-hybrid KBs with weakly DL-safe rules for all the DLs and all the classes of nonrecursive programs above considered. More specifically, the table in Figure 3 summarizes the results on data complexity of NM-satisfiability (as well as for FOL-satisfiability) which hold for the class of r-hybrid KBs with weakly DL-safe rules. The complexity is the same for all the classes of nonrecursive Datalog rules considered in this paper.

A comparison of the table in Figure 3 with the previous one in Figure 2 allows us to evaluate the impact of the weak-DL-safeness assumption on the complexity of reasoning in r-hybrid KBs. Indeed, restricting the interaction between DLs and rules through the weak DL-safeness condition allows for using even very expressive DLs as the ontology language of the r-hybrid KB, without losing decidability of reasoning. In particular,

	NR-Datalog, *NR-Datalog*$^{\neq}$, *NR-Datalog*$^{\neg}$
DL-Lite$_{RDFS}$	$\leq$LOGSPACE
DL-Lite$_{R}$	$\leq$LOGSPACE
$\mathcal{EL}$	= PTIME
from $\mathcal{ALC}$ to $\mathcal{SHIQ}$	= NP
$\mathcal{DLR}$	DECIDABLE, $\geq$ NP

Fig. 3. Data complexity of both NM-satisfiability and FOL-satisfiability in r-hybrid KBs with nonrecursive weakly DL-safe programs

Theorem 6 implies that, under the weak DL-safeness condition, it is possible to combine every DL considered in this paper with full Datalog$^{\neg\vee}$ programs (i.e., with recursion, inequality, negation, and disjunction in the head), and obtain a decidable formalism. Moreover, Figure 3 shows that, for all the DLs and the classes of nonrecursive Datalog rules considered in this paper, when we impose weak DL-safeness the data complexity of reasoning is no worse than the data complexity of reasoning in the absence of rules: i.e., adding weakly DL-safe nonrecursive rules does not actually affect data complexity of reasoning in all the DLs considered.

On the other hand, the unrestricted integration of DLs and rules imposes severe restrictions on the expressive power of both the DL component and the rule component: indeed, as explicitly shown by Figure 2, decidability in the presence of inequality or negation in rules can be regained at the price of restricting both the ontology language to DLs of very little expressiveness and the rule language to extremely limited fragments of Datalog.

6 Conclusions

In this paper we have tried to extend the computational analysis of reasoning in systems integrating Description Logics ontologies and Datalog rules. To this aim, we have considered a group of Description Logics which, from the viewpoint of the expressive power, lie within the range from RDFS to OWL, and thus constitute very important classes of ontology formalisms with respect to Semantic Web applications. Moreover, we have considered disjunctive Datalog and several subclasses of it, with special emphasis on nonrecursive and nondisjunctive fragments.

In our opinion, the results presented in Section 4 clearly show that the unrestricted interaction of DLs and Datalog is computationally very hard even in the absence of recursion in rules. This contrasts with the general opinion that recursion is a necessary feature for rules to rise the undecidability issue in their integration with DL ontologies. So, surprisingly, the various “safeness” restrictions which have been defined to regain decidability in the interaction between DLs and recursive Datalog, appear as necessary restrictions even when rules are not recursive. In this respect, the results in Section 5 further enlarge the class of Description Logics and rules with decidable, restricted integration, and provide a refined computational analysis for the integration of weakly DL-safe rules with the Description Logics considered in this paper.

The present study can be extended in several directions. In our opinion, the most interesting ones are the following:

- the analysis presented in Section 4 should be extended to other very promising tractable DLs recently defined, in particular *Horn*$\mathcal{SHIQ}$ [19], $\mathcal{EL}^{++}$ [2] and *DL-Lite*$_F$ [5];
- the analysis presented in Section 4 should be further extended to classes of disjunctive programs;
- it would be very interesting, for the decidable cases of Figure 2, to provide upper bounds for *non-head-DL-free* programs;
- with respect to the results presented in Section 5, an important open issue is whether it is possible to identify other forms of decidable interaction between DL-KBs and rules, which overcome the expressive limitations of the weak DL-safeness (see [28]). An approach in this direction is presented in [22], which is based on the use of a modal autoepistemic logic, as well as the approach in [8]. Moreover, other interesting approaches have been presented. Some of the most recent ones study the combination of DLs and rules under a different semantic approach [21] or under different restrictions on variables in rules [17].

Acknowledgments. This research has been partially supported by FET project TONES (Thinking ONtologiES), funded by the EU under contract number FP6-7603, by project HYPER, funded by IBM through a Shared University Research (SUR) Award grant, and by MIUR FIRB 2005 project “Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet” (TOCAI.IT).

References

1. http://www.w3.org/TR/rdf-schema/
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI 2005, pp. 364–369 (2005)
3. 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)
4. Berger, R.: The undecidability of the dominoe problem. Mem. Amer. Math. Soc. 66, 1–72 (1966)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proc. of AAAI 2005, pp. 602–607 (2005)
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR 2006 (2006)
7. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of PODS 1998, pp. 149–158 (1998)
8. de Bruijn, J., Eiter, T., Polleres, A., Tompits, H.: Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In: Proc. of IJCAI 2007 (2007)
9. Eiter, T., Gottlob, G., Mannilla, H.: Disjunctive Datalog. ACM Trans. on Database Systems 22(3), 364–418 (1997)

10. 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)
11. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: Proc. of KR 2004, pp. 141–151 (2004)
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)
13. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic SHIQ. In: Proc. of IJCAI 2007, pp. 399–404 (2007)
14. Glimm, B., Horrocks, I., Sattler, U.: Conjunctive query answering for description logics with transitive roles. In: Proc. of DL 2006. Proceedings of CEUR Electronic Workshop (2006), http://ceur-ws.org/Vol-189
15. Glimm, B., Horrocks, I., Sattler, U.: Conjunctive query answering for the description logic SHOIQ. Technical report, University of Manchester, UK (2006)
16. Grau, B.C.: A possible simplification of the semantic web architecture. In: Proc. of the 13th Int. World Wide Web Conf. (WWW 2004), pp. 704–713 (2004)
17. Heymans, S., de Bruijn, J., Predoiu, L., Feier, C., Nieuwenborgh, D.V.: Guarded hybrid knowledge bases. Theory and Practice of Logic Programming 8(3), 411–429 (2008)
18. Horrocks, I., Patel-Schneider, P.F.: A proposal for an OWL rules language. In: Proc. of the 13th Int. World Wide Web Conf. (WWW 2004), pp. 723–731 (2004)
19. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Proc. of IJCAI 2005, pp. 466–471 (2005)
20. Levy, A.Y., Rousset, M.-C.: Combining Horn rules and description logics in CARIN. Artificial Intelligence 104(1–2), 165–209 (1998)
21. Lukasiewicz, T.: A novel combination of answer set programming with description logics for the semantic web. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 384–398. Springer, Heidelberg (2007)
22. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: Proc. of IJCAI 2007 (to appear, 2007)
23. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. J. of Web Semantics 3(1), 41–60 (2005)
24. Ortiz, M.M., Calvanese, D., Eiter, T.: Characterizing data complexity for conjunctive query answering in expressive description logics. In: Proc. of AAAI 2006 (2006)
25. Ortiz de la Fuente, M.M., Calvanese, D., Eiter, T.: Data complexity of answering unions of conjunctive queries in $\mathcal{SHIQ}$. Technical report, Fac.of Computer Science, Free Univ.of Bozen-Bolzano (March 2006), http://www.inf.unibz.it/ calvanese/papers/orti-calv-eite-TR-2006-03.pdf
26. Patel-Schneider, P.F., Hayes, P.J., Horrocks, I., van Harmelen, F.: OWL web ontology language; semantics and abstract syntax. W3C candidate recommendation (November 2002), http://www.w3.org/tr/owl-semantics/
27. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. J. of Web Semantics 3(1), 61–73 (2005)
28. Rosati, R.: DL+log: tight integration of Description Logics and disjunctive Datalog. In: Proc. of KR 2006, pp. 68–78 (2006)
29. Rosati, R.: The limits of querying ontologies. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 164–178. Springer, Heidelberg (2007)

Simulation Subsumption or Déjà vu on the Web

François Bry, Tim Furche, and Benedikt Linse

Institute for Informatics, University of Munich,
Oettingenstraße 67, 80538 München, Germany
`http://pms.ifi.lmu.de/`

Abstract. Simulation unification is a special kind of unification adapted to retrieving semi-structured data on the Web. This article introduces *simulation subsumption,* or containment, that is, query subsumption under simulation unification. Simulation subsumption is crucial in general for query optimization, in particular for optimizing pattern-based search engines, and for the termination of recursive rule-based web languages such as the XML and RDF query language Xcerpt. This paper first motivates and formalizes simulation subsumption. Then, it establishes decidability of simulation subsumption for advanced query patterns featuring descendant constructs, regular expressions, negative subterms (or subterm exclusions), and multiple variable occurrences. Finally, we show that subsumption between two query terms can be decided in $\mathcal{O}(n!^n)$ where n is the sum of the sizes of both query terms.

1 Introduction

Xcerpt query terms [1] are an answer to accessing Web data in a rule-based query language. Like most approaches to Web data (or semi-structured data, in general), they are distinguished from relational query languages such as SQL by a set of query constructs specifically attuned to the less rigid, often diverse, or even entirely schema-less nature of Web data. Xcerpt terms are similar to normalized forward XPath (see [2]) but extended with variables, deep-equal, a notion of injective match, regular expressions, and full negation. Thus, they achieve much of the expressiveness of XQuery without sacrificing the simplicity and pattern-structure of XPath.

When used in the context of Xcerpt, query terms serve a similar role to terms of first-order logic in logic languages. Therefore, the notion of unification has been adapted for Web data in [3], there called "simulation unification". This form of unification is capable of handling all the extensions of query terms over first-order terms that are needed to support Web data: selecting terms at arbitrary depth (`desc`), distinguishing partial from total terms, regular expressions instead of plain labels, negated subterms (`without`), etc.

To illustrate the notion of query term, consider the following query term. It selects the content of title elements at arbitrary depth (`desc`) under a book element in a bibliography database. In addition, we ask for the author of such a book, but only if both first-name and last-name of that author are recorded in that order. Finally, if there is also information about the publisher of that book, we retrieve that information as well:

```
bib{{
  book{{
    desc title{ var Title },
    var Author as author[[ first-name{{ }}, last-name{{ }} ]],
    optional var publisher as publisher{{ }}
  }}
}}
```

Subsumption or containment of two queries (or terms) is an established technique for optimizing query evaluation: a query q_1 is said to be *subsumed* by or *contained* in a query q_2 if every possible answer to q_1 against every possible data is also an answer to q_2. Thus, given all answers to q_2, we can evaluate q_1 only against those answers rather than against the whole database.

For first-order terms, subsumption is efficient and employed for guaranteeing termination in tabling (or memoization) approaches to backward chaining of logic [4,5]. However, when we move from first-order terms to Web queries subsumption (or containment) becomes quickly less efficient or even intractable. Xcerpt query terms have, as pointed out above, some similarity with XPath queries. Containment for various fragments of XPath is surveyed in [6], both in absence and in presence of a DTD. Here, we focus on the first setting, where no additional information about the schema of the data is available. However, Xcerpt query terms are a strict super-set of (navigational) XPath as investigated in [6]. In particular, the Xcerpt query terms may contain (multiple occurrences of the same) variables. This brings them closer to *conjunctive queries* (with negation and deep-equal), as considered in [7] on general relations, and in [8] for tree data. Basic Xcerpt query terms can be reduced to (unions of) conjunctive queries with negation. However, the injectivity of Xcerpt query terms (no two siblings may match with the same data node) and the presence of deep-equal (two nodes are deep-equal iff they have the same structure) have no direct counterpart in conjunctive query containment. Though [9] shows how inequalities in general affect conjunctive query containment, the effect of injectivity (or all-distinct constraints) on query containment has not been studied previously. The same applies to deep-equal, though the results in [10] indicate that in *absence* of composition deep-equal has no effect on evaluation and thus likely on containment complexity.

For Xcerpt query terms, subsumption is, naturally, of interest for the design of a terminating, efficient Xcerpt engine. Beyond that, however, it is particularly relevant in a Web setting. Whenever we know that one query subsumes another, we do not need to access whatever data the two queries access twice, but rather can evaluate both queries with a single access to the basic data by evaluating the second query on the answers of the first one. This can be a key optimization also in the context of search engines, where answers to frequent queries can be memorized so as to avoid their repeated computation. Even though today's search engines are rather blind of the tree or graph structure of HTML, XML and

RDF data, there is no doubt that some more or less limited form of structured queries will become more and more frequent in the future (see Google scholar's "search by author, date, etc."). Query subsumption, or containment, is key to a selection of queries, the answers to which are to be stored so as to allow as many queries as possible to be evaluated against that small set of data rather than against the entire search engine data. Thus, the notion of simulation subsumption proposed in this paper can be seen as a building block of future, structure-aware search engines.

Therefore, we study in this paper subsumption of Xcerpt query terms. To that end, the main contributions are:

- we introduce and formalize a notion of subsumption for Xcerpt query terms, called *simulation subsumption*, in Section 3. To the best of our knowledge, this is the first notion of subsumption for queries with injectivity of sibling nodes and deep-equal.
- we show, also in Section 3, that simulation on query terms is equivalent to simulation subsumption. This also shows that simulation unification as introduced in [3] indeed captures the intuition that a query term that simulates into another query term subsumes that term. Furthermore, all results for simulation subsumption apply equally to simulation unification.
- we define, in Section 4, a *rewriting system* that allows us to reduce the test for subsumption of q in q' to finding a sequence of syntactic transformations that can be applied to q to transform it into q'.
- we show, in Section 5, that this rewriting system gives rise to an algorithm for testing subsumption that is sound and complete and can determine whether q subsumes q' in time $\mathcal{O}(n!^n)$. In particular, this shows that simulation subsumption is decidable.

2 Xcerpt Basics: Query Terms and Simulation

This section lays the foundations for simulation subsumption by introducing the notions of semi-structured trees (Definition 1), query terms (Definition 2) and simulation (Definition 5). Semi-structured trees are an abstraction for all kinds of Web data such as XML-documents, RDF graphs or HTML-documents.

Definition 1 (Semi-structured Trees). *Trees (also called data terms in the following) are are inductively defined as follows:*

- *a label is an atomic tree*
- *if l is a label and $t_1, \ldots, t_n$ are trees with $n \geq 1$, then $l\{t_1, \ldots, t_n\}$ is a tree.*

Query terms are an abstraction for queries that can be used to extract data from semi-structured trees. In contrast to XPath queries, they may contain (multiple occurrences of the same) variables and demand an *injective mapping* of the child terms of each term. For example, the XPath query `/a/b[c]/c` demands that the document root has label `a`, and has a child term with label `b` that has itself a child term with label c. The subterm c that is given within the predicate of b can be mapped to the same node in the data as the child named c of b. Therefore this XPath query would be equivalent to the query term $a\{\{b\{\{c\}\}\}\}$, but not to $a\{\{b\{\{c, c\}\}\}\}$. Simulation could be, however, easily modified to drop the injectivity requirement.

Recall the example query term from Section 1:

```
bib{{
  book{{
    desc title{ var Title },
    var Author as author[[ first-name{{ }}, last-name{{ }} ]],
    optional var publisher as publisher{{ }}
  }}
}}
```

This query term illustrates most of the features of Xcerpt query terms relevant for this paper. As stated above, it selects titles, authors, and optionally publishers of the same book. Titles may occur at any depth under the book element (indicated by desc), but authors and publishers must be children. For authors we further ask that they also record first and last name and that these are recorded in that order (i.e., not last-name before first-name).

Single (double, resp.) braces or brackets in an Xcerpt query term mean that the term's content model is completely (incompletely, resp.) specified (i.e. there must only be a single subterm within the title element of the example from Section 1, but the author element may contain other children besides first-name and last-name). (Curly) braces mean that the order of occurrence of the subterm in the data is irrelevant, (square) brackets enforce the same order in the data as in the query term (i.e. first-name must appear before last-name in the data, otherwise the query term from Section 1 does not match).

Even though there are (many) XML serializations for RDF data, most prominently RDF/XML, none convey the inherent graph structure of RDF data. Each RDF serialization either approximates an RDF graph by a tree, or decomposes it into triples. Xcerpt natively supports RDF with constructs conveying RDF specifics such as containers, collections, the type system and reification. For the sake of focus and simplicity, these RDF constructs are not addressed in the present paper. A complete presentation of Xcerpt's construct for RDF is given in [11].

Definition 2 ((Xcerpt) Query Terms). *Query terms over a set of labels $\mathcal{N}$, a set of variables $\mathcal{V}$, and a set of regular expressions $\mathcal{R}$ are inductively defined as follows:*

- for each label $l \in \mathcal{N}$, l`{{ }}` and l`{ }` are atomic query terms. l is a short hand notation for l`{{ }}`. The formal treatment of square brackets in query terms is omitted in this contribution for the sake of brevity.
- for each variable $X \in \mathcal{V}$, var X is a query term
- for each regular expression $r \in \mathcal{R}$, /r/ is a query term. With $\mathcal{L}(r)$ we denote the set of labels matched by r, i.e. the language defined by the regular expression.

- for each variable $X \in \mathcal{V}$ and query term t, X`as` t is a query term. t is called a *variable restriction* for X.
- for each query term t, `desc` t is a query term and called *depth-incomplete* or *incomplete in depth.*
- for each query term t, `without` t is a query term and called a *negated subterm.*
- for each label l and query terms $t_1, \ldots, t_n$ are query terms with $n \geq 1$,

$$q_1 = l\texttt{\{\{ } t_1\texttt{, \textbackslash ldots, } t_n\texttt{\}\}}$$
$$q_2 = l\texttt{\{ } t_1\texttt{, \textbackslash ldots, } t_n\texttt{\}}$$

  are query terms. q_1 is said to be *incompletely specified in breadth*, or simply *breadth-incomplete*, whereas q_2 is *completely specified in breadth*, or simply *breadth-complete.*

In the following, we let $\mathcal{D}$ and $\mathcal{Q}$ denote the set of all semi-structured trees and query terms, respectively.

A query term and a semi-structured tree are in the simulation relation, if the query term "matches" the data. Matching trees with data is very similar to matching Xpath queries with XML documents – apart from the variables and the injectivity requirement in query terms. The formal definition of simulation of a query term with a semi-structured tree is somewhat involved. To shorten the presentation, we first introduce some notation:

Definition 3 (Injective and Bijective Mappings). *Let* $I := \{t_1^1, \ldots, t_k^1\}$, $J := \{t_1^2, \ldots, t_n^2\}$ *be sets of query terms and* $\pi : I \Rightarrow J$ *be a mapping.*

- π *is* injective*, if all* $t_i^1, t_j^1 \in I$ *satisfy* $t_i^1 \neq t_j^1 \Rightarrow \pi(t_i^1) \neq \pi(t_j^1)$.
- π *is* bijective*, if it is injective and for all* $t_j^2 \in J$ *there is some* $t_i^1 \in I$ *such that* $\pi(t_i^1) = t_j^2$.

We use the following abbreviations to reference parts of a query term q:

$l(q)$: the label of q,
$ChildT(q)$: the set of child subterms of q, i.e. those directly nested inside of q.
$ChildT^+(q)$: the set of positive direct subterms (i.e. those direct subterms which are not of the form *without*...),
$ChildT^-(q)$: the set of negated direct subterms (i.e. the direct subterms of the form *without*...),
$Desc(q)$: the set of direct descendant subterms of q (i.e. those of the from *desc*...),
$SubT(q)$: the direct or indirect subterms of q, i.e. all direct subterms as well as their subterms.
$ss(q)$: the subterm specification of q. It can either be *complete* (single curly braces) or *incomplete* (double curly braces).
$vars(q)$: the set of variables occurring somewhere in q.
$pos(q)$: q', if q is of the form `without` q' for some query term, q otherwise.

Definition 4 (Ground Query Term Simulation). *Let q be a query term and d be a semi-structured tree, A relation $\mathcal{S} \subseteq (SubT(q) \cup \{q\}) \times (SubT(d) \cup \{d\})$ is a simulation of q into d if the following holds:*

- $q \; \mathcal{S} \; d$
- *if $q := l_1\{\{q_1, \ldots, q_n\}\} \; \mathcal{S} \; l_2\{d_1, \ldots, d_m\} =: d$ then l_1 must subsume l_2, and there must be an injective mapping $\pi : ChildT^+(q) \to ChildT^+(d)$ such that $q_i \; \mathcal{S} \; \pi(q_i)$ for all $i \in ChildT^+(q)$. Moreover, there* must not *be a $q_j \in ChildT^-(q)$ and $d_l \in ChildT^+(d) \setminus range(\pi)$ such that $pos(q_j) \; \mathcal{S} \; d_l$.*
- *if $q := l1\{q_1, \ldots, q_n\} \; \mathcal{S} \; l2\{d_1, \ldots, d_m\} =: d$ then l_1 must subsume l_2, and there must be a bijective mapping $\pi : ChildT^+(q) \to ChildT^+(d)$ such that $q_i \; \mathcal{S} \; \pi(q_i)$ for all $i \in ChildT^+(q)$. Note that the set $ChildT^-(q)$ of negated direct subterms of q should be empty – the presence of negated subterms in breadth-complete query terms is irrelevant.*
- *if $q = desc \; q' \; \mathcal{S} \; d$ then $q' \; \mathcal{S} \; d$ or $q' \; \mathcal{S} \; d'$ for some subterm d' of d.*

If there is a relation $\mathcal{S}$ that satisfies the above conditions, q simulates into *d (short: $q \preceq d$; to state the contrary we write $q \not\preceq d$).*

Since every semi-structured tree is also a query term, the above definition of simulation between a query term and a tree can be extended to a relation between pairs of query terms. For the sake of brevity this full definition of *extended ground query term simulation* is given in the appendix of the online version [12].

The existence of a ground query term simulation states that a given semi-structured tree satisfies the conditions encapsulated in the query term. Many times, however, query authors are not only interested in checking the structure and content of a document, but also in extracting data from the document, and therefore query terms may contain logical variables. To formally specify the data that is extracted by matching a query term with a semi-structured tree, non-ground query term is introduced (Definition 5). Substitutions are defined as usual, and the application of a substitution to a query term is the consistent replacement of the variables by their images in the substitution.

Definition 5 (Non-Ground Query Term Simulation). *A query term q with variables simulates into a semi-structured tree d iff there is a substitution $\sigma : Vars(q) \to \mathcal{D}$ such that $q\sigma$ simulates into d.*

3 Simulation Subsumption

In this section, we first introduce simulation subsumption (Definition 6), then for several query terms we discuss whether one subsumes the other to give an intuition for the compositionality of the subsumption relationship. Subsequently, the transitivity of the subsumption relationship is proven (Lemma 1), some conclusions about the membership in the subsumption relationship of subterms, given the membership in the subsumption relationship of their parent terms are stated. These conclusions formalize the compositionality of simulation subsumption and are a necessary condition for the completeness of the rewriting system introduced in Section 4.

In tabled evaluation of logic programs, solutions to subgoals are saved in a solution table, such that for equivalent or subsumed subgoals, these sets do not have to be recomputed. As mentioned before, this avoidance of re-computation does not only save time, but can, in certain cases be crucial for the termination of a backward chaining evaluation of a program. In order to classify subgoal as solution or look-up goals, boolean subsumption as specified by Definition 6 must be decided. Although Xcerpt query terms may contain variables, n-ary subsumption as defined in [6] would be too strict for our purposes. To see this, consider the Xcerpt query terms $q_1 := a\{\{var\ X\}\}$ and $q_2 := a\{\{c\}\}$. Although all trees that are relevant for q_2 can be found in the solutions for q_1, q_1 and q_2 cannot be compared by n-ary containment, because they differ in the number of their query variables.

Definition 6 (Simulation Subsumption). *A query term q_1* subsumes *another query term q_2 if all data or query terms that q_2 simulates with are also simulated by q_1.*

Example 1 (Examples for the subsumption relationship). Let $q_1 := a\{\{\}\}$, $q_2 := a\{\{desc\ b, desc\ c, d\}\}$, $q_3 := a\{\{desc\ b, c, d\}\}$, $q_4 := a\{\{without\ e\}\}$, and $q_5 := a\{\{without\ e\{\{without\ f\}\}\}\}$. Then the following subsumption relationships hold:

- q_2 subsumes q_3 because it requires less than q_3: While q_3 requires that the data has outermost label a, subterms c and d as well as a descendant subterm b, q_2 requires not that there is a direct subterm c, but only a descendant subterm. Since every descendant subterm is also a direct subterm, all trees simulating with q_3 also simulate with q_2.
  But the subsumption relationship can also be decided in terms of simulation: q_2 subsumes q_3, because there is a mapping π from the direct subterms $Child(q_2)$ of q_2 to the direct subterms $Child(q_3)$ of q_3, such that q_i subsumes $\pi(q_i)$ for all q_i in $Child(q_2)$.
- q_3 does not subsume q_2, since there are trees that simulate with q_2, but not with q_3. One such tree is $d := a\{b, e\{c\}, d\}$.
  Again, the subsumption relationship between q_3 and q_2 (in this order) can be decided by simulation. There is no mapping π from the direct subterms of q_3 to the direct subterms of q_2, such that a simulates into $\pi(a)$.
- q_1 subsumes q_4 since it requires less than q_4. All trees that simulate with q_4 also simulate with q_1.
- q_4 does not subsume q_1, since the tree $a\{\{e\}\}$ simulates with q_1, but does not simulate with q_4.
- q_5 subsumes q_4, but not the other way around.

Proposition 1. *The subsumption relationship between query terms is transitive, i.e. for arbitrary query terms q_1, q_2 and q_3 it holds that if q_1 subsumes q_2 and q_2 subsumes q_3, then q_1 subsumes q_3.*

Proposition 1 immediately follows from the transitivity of the subset relationship. Query term simulation and subsumption are defined in a way such that, given the simulation subsumption between two query terms, one can draw conclusions about subsumption relationships that must be fulfilled between pairs of subterms of the query terms. Lemma 1 formalizes these sets of conclusions.

Lemma 1 (Subterm Subsumption). *Let q_1 and q_2 be query terms such that q_1 subsumes q_2. Then there is an injective mapping π from $Child^+(q_1)$ to $Child^+(q_2)$ such that q_1^i subsumes $\pi(q_1^i)$ for all $q_1^i \in Child^+(q_1)$.*

Furthermore, if q_1 and q_2 are breadth-incomplete, then there is a (not necessarily injective) mapping σ from $Child^-(q_1)$ to $Child^-(q_2)$ such that $pos(\sigma(q_1^j))$ subsumes $pos(q_1^j)$ for all $q_1^j \in Child^-(q_1)$.

If q_1 is breadth-incomplete and q_2 is breadth-complete then there is no q_1^j in $Child^-(q_1)$ and $q_2^k \in Child^+(q_2) \setminus range(\pi)$ such that $pos(q_1^j) \preceq q_2^k$.

Lemma 1 immediately follows from the equivalence of the subsumption relationship and the extended query term simulation (see Lemma 4 in the appendix of the online version [12]).

4 Simulation Subsumption by Rewriting

In this section we lay the foundations for a proof for the decidability of subsumption between query terms according to Definition 6 by introducing a rewriting system from one query term to another, which is later shown to be sound and complete. Furthermore this rewriting system lays the foundation for the complexity analysis in Section 5.3.

The transformation of a query term q_1 into a subsumed query term q_2 is exemplified in Figure 1.

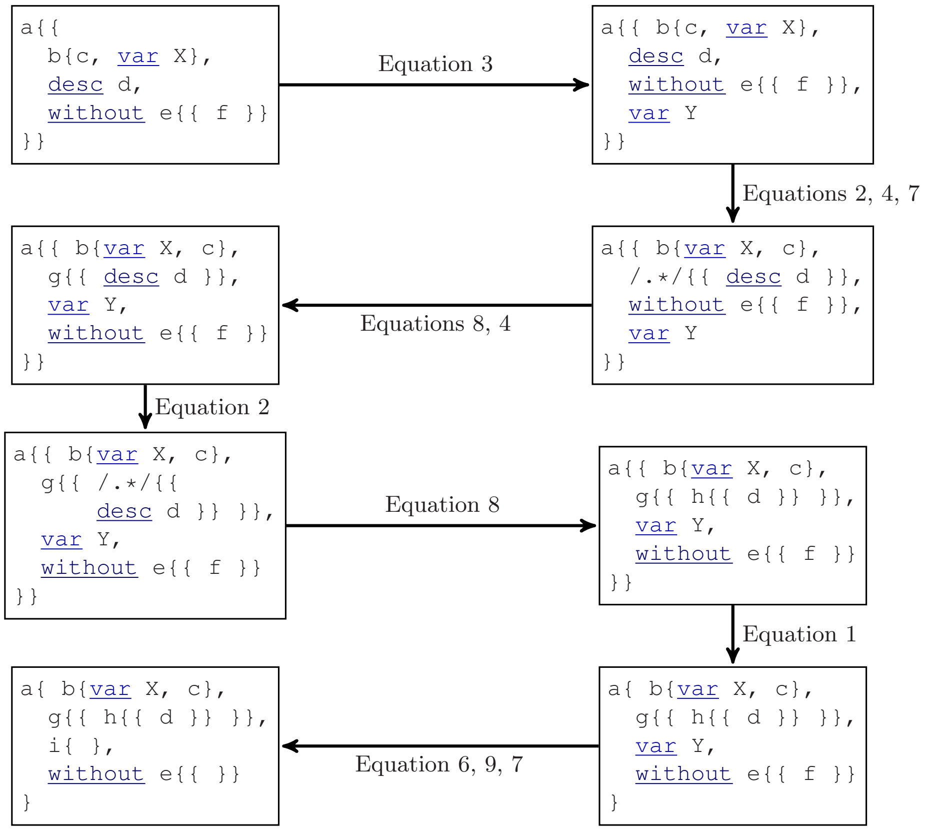
Definition 7 (Subsumption Monotone Query Term Transformations). *Let q be a query term. The following is a list of so-called* subsumption monotone *query term transformations.*

- *if q has incomplete subterm specification, it may be transformed to the analogous query term with complete subterm specification.*

$$\frac{a\{\{q_1,\ldots,q_n\}\}}{a\{q_1,\ldots,q_n\}}, \tag{1}$$

- *if q is of the form desc q' then the descendant construct may be eliminated or it may be split into two descendant constructs separated by the regular expression `/.*/`, the inner descendant construct being wrapped in double curly braces.*

$$\frac{desc\ q}{q}, \qquad \frac{desc\ q}{desc\ /.*/\{\{desc\ q\}\}} \tag{2}$$



```
a{{
  b{c, var X},
  desc d,
  without e{{ f }}
}}
```

```
a{{ b{c, var X},
  desc d,
  without e{{ f }},
  var Y
}}
```

```
a{{ b{var X, c},
  /.*/{{ desc d }},
  without e{{ f }},
  var Y
}}
```

```
a{{ b{var X, c},
  g{{ desc d }},
  var Y,
  without e{{ f }}
}}
```

```
a{{ b{var X, c},
  g{{ /.*/{{
      desc d }} }},
  var Y,
  without e{{ f }}
}}
```

```
a{{ b{var X, c},
  g{{ h{{ d }} }},
  var Y,
  without e{{ f }}
}}
```

```
a{ b{var X, c},
  g{{ h{{ d }} }},
  var Y,
  without e{{ f }}
}
```

```
a{ b{var X, c},
  g{{ h{{ d }} }},
  i{ },
  without e{{ }}
}
```

Fig. 1.

– *if q has incomplete-unordered subterm specification, then a fresh variable X may be appended to the end of the subterm list. A* fresh *variable is a variable that does not occur in q_1 or q_2 and is not otherwise introduced by the rewriting system.*

$$X \text{ fresh} \Rightarrow \frac{a\{\{q_1, \ldots, q_n\}\},}{a\{\{q_1, \ldots, q_n, var\ X\}\}} \tag{3}$$

– *if q has unordered subterm specification, then the subterms of q may be arbitrarily permuted.*

$$\pi \in Perms(\{1, \ldots, n\}) \Rightarrow \frac{a\{\{q_1,\ \ldots,\ q_n\}\}}{a\{\{q_{\pi(1)},\ \ldots,\ q_{\pi(n)}\}\}} \tag{4}$$

$$\pi \in Perms(\{1, \ldots, n\}) \Rightarrow \frac{a\{q_1,\ \ldots,\ q_n\}}{a\{q_{\pi(1)},\ \ldots,\ q_{\pi(n)}\}} \tag{5}$$

– *if q contains a variable* `var X`*, which occurs in q at least once in a positive context (i.e. not within the scope of a without) then all occurrences of* `var X` *may be substituted by another Xcerpt query term.*

$$X \in PV(q), t \in QTerms \Rightarrow \frac{q}{q\{X \mapsto t\}} \tag{6}$$

This rule may only be applied, if q contains all *occurrences of X in q_1. Furthermore, no further rewriting rules may be applied to the replacement term t.*

Notice that if a variable appears within q only in a negative context (i.e. within the scope of a `without`*), the variable cannot be substituted by an arbitrary term to yield a transformed term that is subsumed by q. The query terms* `a{{ without var X }}` *and* `a{{ without b{ } }}` *together with the tree* `a{ c }` *illustrate this characteristic of the subsumption relationship. For further discussion of substitution of variables in a negative context see Example 2.*

– *if q has a subterm q_i, then q_i may be transformed by any of the transformations in this list except for Equation 6 to the term $t(q_i)$, and this transformed version may be substituted at the place of q_i in q, as formalized by the following rule:* [1][2]

$$\frac{q_i}{t(q_i)} \Rightarrow \frac{a\{\{q_1, \ldots, q_n\}\}}{a\{\{q_1, \ldots, q_{i-1}, t(q_i), q_{i+1}, \ldots q_n\}\}} \tag{7}$$

– *if the label of q is a regular expression e, this regular expression may be replaced by any label that matches with e, or any other regular expression e' which is subsumed by e (see Definition 8 in the appendix of the online version [12]).*[1]

$$e \in RE, e \text{ subsumes } e' \Rightarrow \frac{e\{\{q_1, \ldots, q_n\}\}}{e'\{\{q_1, \ldots, q_n\}\}} \tag{8}$$

– *if q contains a negated subterm $q_i =$ without r and r' is a query term such that $t(r') = r$ (i.e. r' subsumes r) for some transformation step t, then q_i can be replaced by $q_i' :=$ without r'.*

$$(q_i = without\ r) \wedge \frac{r'}{r} \wedge (q_i' = without\ r') \Rightarrow \frac{a\{\{q_1, \ldots, q_i, \ldots, q_n\}\}}{a\{\{q_1, \ldots, q_i', \ldots q_n\}\}} \tag{9}$$

[1] The respective rules for complete-unordered subterm specification, incomplete-ordered subterm specification and complete-ordered subterm specification are omitted for the sake of brevity.

[2] The exclusion of Equation 6 ensures that variable substitutions are only applied to entire query terms and not to subterms. Otherwise the same variable might be substituted by different terms in different subterms.

5 Properties of the Rewriting System

In this section we show that the rewriting system introduced in the previous section is sound (Section 5.1) and complete (Section 5.2). Furthermore we study the structure of the search tree induced by the rewriting rules, show that it can be pruned without losing the completeness of the rewriting system and conclude that simulation subsumption is decidable. Finally we derive complexity results from the size of the search tree in Section 5.3.

5.1 Subsumption Monotonicity and Soundness

Lemma 2 (Monotonicity of the Transformations in Definition 7). *All of the transformations given in Definition 7 are subsumption monotone, i.e. for any query term q and a transformation from Definition 7 which is applicable to q, q subsumes $t(q)$.*

The proof of Lemma 2 is straight-forward since each of the transformation steps can be shown independently of the others. For all of the transformations, inverse transformation steps t^{-1} can be defined, and obviously for any query term q it holds that $t^{-1}(q)$ subsumes q.

Lemma 3 (Transitivity of the Subsumption Relationship, Monotonicity of a Sequence of Subsumption Monotone Query Term Transformations). *For a sequence of subsumption monotone query term transformations $t_1, \ldots, t_n$, and an arbitrary query term q, q subsumes $t_1 \circ \ldots \circ t_n(q_1)$.*

The transitivity of the subsumption relationship is immediate from its definition (Definition 6) which is based on the subset relationship, which is itself transitive.

As mentioned above, the substitution of a variable X in a negative context of a query term q by a query term t, which is not a variable, results in a query term $q' := q[X \mapsto t]$ which is in fact more general than q. In other words $q[X \mapsto t]$ subsumes q for any query term q if X only appears within a negative context in q. On the other hand, if X only appears in a positive context within q, then q' is less general – i.e. q subsumes q'. But what about the case of X appearing both in a positive and a negative context within q? Consider the following example:

Example 2. Let $q :=$ `a{{ var X, without b{{ var X }} }}`. One may be tempted to think that substituting X by `c[]` to give q' makes the first subterm of q less general, but the second subterm of q more general. In fact, a subterm `b[ c ]` within a tree would cause the subterm `without b{{ var X }}` of q to fail, but the respective subterm of q' to succeed, suggesting that there is a tree that simulation unifies with q', but not with q, meaning that q does not subsume q'. However, there is no such tree, which is due to the fact that the second occurrence of `X` within q is only a consuming occurrence. When this part of the query term is evaluated, the variable `X` is already bound.

The normalized form for Xcerpt query terms is introduced, because for an unnormalized query term q_1 that subsumes a query term q_2 one cannot guarantee that there is a sequence of subsumption monotone query term transformations $t_1, \dots, t_n$ such that $t_n \circ \dots \circ t_1(q_1) = q_2$. To see this, consider example 3.

Example 3 (Impossibility of transforming an unnormalized query term). Let $q_1 := a\{\{var\ X\ as\ b\{\{c\}\}, var\ X\ as\ b\{\{d\}\}\}\}$ and $q_2 := a\{\{b\{\{c,d\}\}, b\{\{c,d\}\}\}\}$. q_2 subsumes q_1, in fact both terms are even simulation equivalent. But there is no sequence of subsumption monotone query term transformations from q_2 to q_1, since one would have to omit one subterm from both the first subterm of q_2 and from the second one. But such a transformation would not be subsumption monotone.

To overcome this issue, query terms are assumed to be in normalized form (Definition 9 in the appendix of the online version [12]). In fact, almost all Xcerpt query terms can be transformed into normalized form.

5.2 Completeness

Theorem 1 (Subsumption by Transformation). *Let q_1 and q_2 be two query terms in normalized form such that q_1 subsumes q_2. Then q_1 can be transformed into q_2 by a sequence of subsumption monotone query term transformations listed in Definition 7.*

Proof. We distinguish two cases:

– q_1 and q_2 are subsumption equivalent (i.e. they subsume each other)
– q_1 strictly subsumes q_2

The first case is the easier one. If q_1 and q_2 are subsumption equivalent, then there is no tree t, such that t simulates with one, but not the other. Hence q_1 and q_2 are merely syntactical variants of each other. Then q_1 can be transformed into q_2 by consistent renaming of variables (Equation 7), and by reordering sibling terms within subterms of q (Equation 4). Note that this would not be true for unnormalized query terms as Example 3 shows.

The second is shown by structural induction on q_1.

For both the induction base and the induction step, we assume that q_1 subsumes q_2, but not the other way around. Then there is a tree d, such that q_1 simulates into d, but q_2 does not. In both the induction base and the induction step, we give a distinction of cases, enumerating all possible reasons for q_1 simulating into d but q_2 not. For each of these cases, a sequence of subsumption monotone transformations $t_1, \dots t_n$ from Definition 7 is given, such that $q_1' := t_n \circ t_{n-1} \circ \dots \circ t_1(q_1)$ does *not* simulate into d. By Lemmas 2 and 3, q_1' still subsumes q_2. Hence by considering d and by applying the transformations, q_1 is brought “closer” to q_2. If q_1' is still more general than q_2, then one more dataterm d' can be found that simulates with q_1', but not with q_2, and another sequence of transformations to be applied can be deduced from this theorem. This process can be repeated until q_1 has been transformed into a simulation equivalent version of q_2. For the proof see the appendix of the online version [12].

5.3 Decidability and Complexity

In the previous section, we establish that, for each pair of query terms q_1, q_2 such that q_1 subsumes q_2, there is a (possibly infinite) sequence of transformations $t_1, \ldots, t_k$ by one of the rules in Section 4 such that $t_k \circ \ldots \circ t_1(q) = q_2$.

However, if we reconsider the proof of Theorem 1, it is quite obvious that the sequence of transformations can in fact not be infinite: Intuitively, we transform at each step in the proof q_1 further towards q_2, guided by a data term that simulates in q_1 but not in q_2. In fact, the length of a transformation sequence is bounded by the sum of the sizes of the two query terms. As size of a query term we consider the total number of its subterms.

Proposition 2 (Length of Transformation Sequences). *Let q_1 and q_2 be two Xcerpt query terms such that q_1 subsumes q_2 and n the sum of the sizes of q_1 and q_2. Then, there is a sequence of transformations $t_1, \ldots, t_k$ such that $t_k \circ \ldots \circ t_1(q_1) = q_2$ and $k \in \mathcal{O}(n)$.*

Proof. We show that the sequences of transformations created by the proof of Theorem 1 can be bounded by $\mathcal{O}(n + m)$ if computed in a specific way: We maintain a mapping μ from subterms of q_1 to subterms of q_2 indicating how the query terms are mapped. μ is initialized with (q_1, q_2). In the following, we call a data term d *discriminating* between q_1 and q_2 if q_1 simulates in d but not q_2.

(1) For each pair (q, q') in μ, we first choose a discriminating data term that matches case 1 in the proof of Theorem 1. If there is such a data term, we apply Equation (8), label replacement, once to q obtaining $t(q)$ and update the pair in μ by $(t(q), q')$. This step is performed at most once for each pair as $(t(q), q')$ have the same label and thus there is no more discriminating data term that matches case 1.

(2) Otherwise, we next choose a discriminating data term that matches case 2.a.i or 2.b.i. In both cases, we apply Equation (3), variable insertion, to insert a new variable and update the pair in μ. This step is performed at most $|q_2| - |q_1| \leq n$ times for each pair.

(3) Otherwise, we next choose a discriminating data term that matches case 2.a.ii and apply Equation (1), complete term specification and update the pair in μ. This step is performed at most once for each pair.

(4) Finally, the only type of discriminating data term that remains is one with the same number of positive child terms as q_2. We use an oracle to guess the right mapping σ from child terms of q_1 to child terms of q_2. Then we remove the pair from μ and add $(c, \sigma(c))$ to μ for each child term of q_1. This step is performed at most once for each pair in μ.

Since query subterms have a single parent, we add each subterm only once to μ in a pair. Except for case 2, we perform only a constant number of transformations to each pair. Case 2 allows up to n transformations for a single pair, but the total number of transformations (over all pairs) due to case 2 is bound by the size of q_2. Thus in total we perform at most $4 \cdot n$ transformations where n is the sum of the number of the sizes of q_1 and q_2.

Though we have established that the length of a transformation sequence is bound by $\mathcal{O}(n)$, we also have to consider how to *find* such a transformation sequence. The proof of Proposition 2, already spells out an algorithm for finding such transformation sequences. However, it uses an oracle to guess the right mapping between child terms of two terms that are to be transformed. A naive deterministic algorithm needs to consider all possible such mappings whose number is bound by $\mathcal{O}(n!)$. It is worth noting, however, that in most cases the actual number of such mappings is much smaller as most query terms have fairly low breadth and the possible mappings between their child terms are severely reduced just by considering only mappings where the labels of child terms simulate. However, in the worst case the $\mathcal{O}(n!)$ complexity for finding the right mapping may be reached and thus we obtain:

Theorem 2 (Complexity of Subsumption by Rewriting). *Let q_1 and q_2 be two Xcerpt query terms. Then we can test whether q_1 subsumes q_2 in $\mathcal{O}(n!^n)$ time.*

Proof. By proposition 2 we can find a $\mathcal{O}(n)$ length transformation sequence in $\mathcal{O}(n!^n)$ time and by Theorem 1 q_1 subsumes q_2 if and only if there is such a sequence.

6 Conclusion

Starting out from the problem of improving termination of logic programming based on rich kinds of simulation such as simulation unification, the problem of deciding simulation subsumption between query terms is investigated in this paper. A rewriting system consisting of subsumption monotone query term transformations is introduced and shown to be sound and complete. By convenient pruning of the search tree defined by this rewriting system, the decidability of simulation subsumption is proven, and an upper bound for its complexity is identified.

Future work includes (a) a proof of concept implementation of the rewriting system, (b) the development of heuristics and their incorporation into the prototype to ensure fast termination of the algorithm in the cases when it is possible, (c) the study of the complexity of the problem in absence of subterm negation, descendant constructs, deep-equal, and/or injectivity, (d) the implementation of a backward chaining algorithm with tabling, which uses subsumption checking to avoid redundant computations and infinite branches in the resolution tree, and (e) the adaptation of the rewriting system to XPath in order to decide subsumption and to derive complexity results for the subsumption problem between XPath queries.

References

1. Schaffert, S., Bry, F.: Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In: Proc. Extreme Markup Languages (Int'l. Conf. on Markup Theory & Practice) (2004)
2. Olteanu, D., Meuss, H., Furche, T., Bry, F.: XPath: Looking forward. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) EDBT 2002. LNCS, vol. 2490. Springer, Heidelberg (2002)
3. Schaffert, S.: Xcerpt: A Rule-Based Query and Transformation Language for the Web. PhD thesis, University of Munich (2004)
4. Tamaki, H., Sato, T.: OLD resolution with tabulation. In: Shapiro, E. (ed.) ICLP 1986. LNCS, vol. 225, pp. 84–98. Springer, Heidelberg (1986)
5. Chen, W., Warren, D.S.: Tabled evaluation with delaying for general logic programs. Journal of the ACM 43(1), 20–74 (1996)
6. Schwentick, T.: XPath query containment. SIGMOD Record 33(1), 101–109 (2004)
7. Wei, F., Lausen, G.: Containment of conjunctive queries with safe negation. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 343–357. Springer, Heidelberg (2002)
8. Björklund, H., Martens, W., Schwentick, T.: Conjunctive Query Containment over Trees. In: Arenas, M., Schwartzbach, M.I. (eds.) DBPL 2007. LNCS, vol. 4797, pp. 66–80. Springer, Heidelberg (2007)
9. Klug, A.: On Conjunctive Queries containing Inequalities. Journal of the ACM 35(1), 146–160 (1988)
10. Koch, C.: On the Complexity of Nonrecursive XQuery and Functional Query Languages on Complex Values. Transactions on Database Systems 31(4) (2006)
11. Pohl, A.: RDF Querying in Xcerpt: Language Constructs and Implementation. Deliverable I4-Dx2, REWERSE (2008)
12. Bry, F., Furche, T., Linse, B.: Simulation subsumption or déjà vu on the web (extended version), http://www.pms.ifi.lmu.de/mitarbeiter/linse/bry-simulation.pdf

Reasoning with a Network of Aligned Ontologies*

Antoine Zimmermann and Chan Le Duc

INRIA Rhône-Alpes - LIG

Abstract. In the context of the Semantic Web or semantic peer to peer systems, many ontologies may exist and be developed independently. Ontology alignments help integrating, mediating or reasoning with a system of networked ontologies. Though different formalisms have already been defined to reason with such systems, they do not consider ontology alignments as first class objects designed by third party ontology matching systems. Correspondences between ontologies are often asserted from an external point of view encompassing both ontologies. We study consistency checking in a network of aligned ontologies represented in Integrated Distributed Description Logics (IDDL). This formalism treats local knowledge (ontologies) and global knowledge (inter-ontology semantic relations, *i.e.*, alignments) separately by distinguishing local interpretations and global interpretation so that local systems do not need to directly connect to each other. We consequently devise a correct and complete algorithm which, although being far from tractacle, has interesting properties: it is independent from the local logics expressing ontologies by encapsulating local reasoners. This shows that consistency of a IDDL system is decidable whenever consistency of the local logics is decidable. Moreover, the expressiveness of local logics does not need to be known as long as local reasoners can handle at least $\mathcal{ALC}$.

1 Introduction

Reasoning on a network of multiple ontologies can be achieved by integration of several knowledge bases or by using non standard distributed logic formalisms. With the first, knowledge must be translated into a common logic, and reasoning is fully centralized. The second option, which has been chosen for Distributed Description Logics (DDL) [3], $\mathcal{E}$-connections [7], Package-based Description Logics (P-DL) [2] or [8] consists in defining new formalisms which allow reasoning with multiple domains in a distributed way. The non-standard semantics of these formalisms reduces conflicts between ontologies, but they do not adequately formalize the quite common case of ontologies related with ontology alignments produced by third party ontology matchers. Indeed, these formalisms assert cross-ontology correspondences (bridge rules, links or imports) from one ontology's point of view, while often, such correspondences are expressed from a point of view that encompasses both aligned ontologies. Consequently, correspondences, being tied to one "context", are not transitive, and therefore, alignments cannot be composed in these languages.

* This work has been partly supported by the European integrated project NeOn (IST-2004-507482).

Our proposed formalism, Integrated Distributed Description Logics (IDDL), addresses this situation and offer sound alignment composition. The principle behind it was presented in [12] under the name *integrated distributed semantics*, and particularized for Description Logics in [10]. This article aims at providing a distributed reasoning procedure for IDDL, which has the following interesting characteristics:

- the distributed process takes advantage of existing DL reasoners (*e.g.*, Pellet, Racer, FacT++, *etc.*);
- local ontologies are encapsulated in the local reasoning system, so it is not necessary to access the content of the ontologies in order to determine the consistency of the overall system;
- the expressiveness of local ontologies is not limited as long as it is a decidable description logic.

The noticeable drawbacks are the following:

- cross-ontology correspondences are limited to concept subsumption or disjointness, and role subsumption (so, role disjointness is not supported); individual correspondences are treated via nominal concepts;
- the algorithm is highly intractable. However this paper is above all concerned by the decidability of IDDL.

The presentation is organized as follows. We start with a presentation of the formalism itself. Then, we describe the reasoning process in the case of concept correspondences alone, in order to lighten the complexity of the notations which will serve to prove the correctness of the algorithm. The following section updates the notations and theorem in the more general case of possible cross-ontology role subsumption. Finally, a discussion on planned implementation and further work is given as well as concluding remarks. Additionally, an Appendix provides a sketch of the proof of the correctness of IDDL decision procedure.

2 Integrated Distributed Description Logics

IDDL is a formalism which inherits from both the field of Description Logics and from the analysis of the forms of distributed semantics in [12].

In a preliminary section, we provide definitions if the syntax and semantics of classical description logics. Thereafter, we provide the definition of correspondence, alignment and distributed system, for which we define a semantics.

2.1 DL: Syntax and Semantics

IDDL ontologies have the same syntax and semantics as in standard DLs. More precisely, a DL ontology is composed of concepts, roles and individuals, as well as axioms built out of these elements. A concept is either a primitive concept A, or, given concepts C, D, role R, individuals $a_1, \dots, a_k$, and natural number n, $\bot$, $\top$, $C \sqcup D$, $C \sqcap D$, $\exists R.C$, $\forall R.C$, $\leq nR.C$, $\geq nR.C$, $\neg C$ or $\{a_1, \dots, a_k\}$. A role is either a primitive role P, or, given roles R and S, $R \sqcup S$, $R \sqcap S$, $\neg R$, R^-, $R \circ S$ and R^+.

Interpretations are pairs $\langle \Delta^I, \cdot^I \rangle$, where Δ^I is a non-empty set (the domain of interpretation) and $\cdot^I$ is the function of interpretation such that for all primitive concepts A, $A^I \subseteq \Delta^I$, for all primitive roles P, $P^I \subseteq \Delta^I \times \Delta^I$, and for all individuals a, $a^I \in \Delta^I$. Interpretations of complex concepts and roles are inductively defined by $\bot^I = \emptyset$, $\top^I = \Delta^I$, $(C \sqcup D)^I = C^I \cup D^I$, $(C \sqcap D)^I = C^I \cap D^I$, $(\exists R.C)^I = \{x | \exists y. y \in C^I \wedge \langle x, y \rangle \in R^I\}$, $(\forall R.C)^I = \{x | \forall y. \langle x, y \rangle \in R^I \Rightarrow y \in C^I\}$, $(\leq nR.C)^I = \{x | \sharp\{y \in C^I | \langle x, y \rangle \in R^I\} \leq n\}$, $(\geq nR.C)^I = \{x | \sharp\{y \in C^I | \langle x, y \rangle \in R^I\} \geq n\}$, $(\neg C)^I = \Delta^I \setminus C^I$, $\{a_1, \ldots, a_k\} = \{a_1^I, \ldots, a_k^I\}$, $(R \sqcup S)^I = R^I \cup S^I$, $(R \sqcap S)^I = R^I \cap S^I$, $(\neg R)^I = (\Delta^I \times \Delta^I) \setminus R^I$, $(R^-)^I = \{\langle x, y \rangle | \langle y, x \rangle \in R^I\}$, $(R \circ S)^I = \{\langle x, y \rangle | \exists z. \langle x, z \rangle \in R^I \wedge \langle z, y \rangle \in S^I\}$ and $(R^+)^I$ is the reflexive-transitive closure of R^I.

Axioms are either subsumptions $C \sqsubseteq D$, sub-role axioms $R \sqsubseteq S$, instance assertions $C(a)$, role assertions $R(a, b)$ and individual identities $a = b$, where C and D are concepts, R and S are roles, and a and b are individuals. An interpretation I *satisfies* axiom $C \sqsubseteq D$ iff $C^I \subseteq D^I$; it satisfies $R \sqsubseteq S$ iff $R^I \subseteq S^I$; it satisfies $C(a)$ iff $a^I \in C^I$; it satisfies $R(a, b)$ iff $\langle a^I, b^I \rangle \in R^I$; and it satisfies $a = b$ iff $a^I = b^I$. When I satisfies an axiom α, it is denoted by $I \models \alpha$.

An ontology O is composed of a set of terms (primitive concepts/roles and individuals) called the signature of O and denoted by $\mathrm{Sig}(O)$, and a set of axioms denoted by $\mathrm{Ax}(O)$. An interpretation I is a model of an ontology O iff for all $\alpha \in \mathrm{Ax}(O)$, $I \models \alpha$. In this case, we write $I \models O$. The set of all models of an ontology O is denoted by $\mathrm{Mod}(O)$. A semantic consequence of an ontology O is a formula α such that for all $I \in \mathrm{Mod}(O)$, $I \models \alpha$.

2.2 Distributed Systems

A distributed system (DS) is composed of a set of ontologies, connected by ontology alignments. An ontology alignment describes semantic relations between ontologies.

Syntax. An ontology alignment is a set of *correspondences*. A correspondence can be seen as an axiom that asserts a relation between concepts, roles or individuals of two distinct ontologies. They are homologous to bridge rules in DDL. We use a notation similar to DDL in order to identify in which ontology a concept, role or individual is defined. If a concept/role/individual E belongs to ontology i, then we write it $i{:}E$. The 6 possible types of correspondences between ontologies i and j are:

Definition 1 (Correspondence). *A* correspondence *between two ontologies i and j is one of the following formulas:*

- *$i{:}C \xleftrightarrow{\sqsubseteq} j{:}D$ is a cross-ontology concept subsumption;*
- *$i{:}R \xleftrightarrow{\sqsubseteq} j{:}S$ is a cross-ontology role subsumption;*
- *$i{:}C \xleftrightarrow{\perp} j{:}D$ is a cross-ontology concept disjointness;*
- *$i{:}R \xleftrightarrow{\perp} j{:}S$ is a cross-ontology role disjointness;*
- *$i{:}a \xleftrightarrow{\in} j{:}C$ is a cross-ontology membership;*
- *$i{:}a \xleftrightarrow{=} j{:}b$ is a cross-ontology identity.*

Notice that it is possible that $i = j$. Ontology alignments and DL ontologies form the components of a Distributed System in IDDL.

Definition 2 (Distributed System). *A* distributed system *or DS is a pair* $\langle \mathbf{O}, \mathbf{A} \rangle$ *such that* $\mathbf{O}$ *is a set of ontologies, and* $\mathbf{A} = (A_{ij})_{i,j \in \mathbf{O}}$ *is a family of alignments relating ontologies of* $\mathbf{O}$.[1]

Semantics. Distributed systems semantics depends on local semantics, but does not interfere with it. A standard DL ontology can be straightforwardly used in IDDL system. Informally, interpreting an IDDL system consists in assigning a standard DL interpretation to each local ontology, then correlating the domains of interpretation thanks to what we call an *equalizing function*.

Definition 3 (Equalizing Function). *Given a family of local interpretations* $\mathbf{I}$, *an equalizing function* $\boldsymbol{\varepsilon}$ *is a family of functions indexed by* $\mathbf{I}$ *such that for all* $I_i \in \mathbf{I}$, $\varepsilon_i : \Delta^{I_i} \rightarrow \Delta_\varepsilon$ *where* Δ_ε *is called the global domain of interpretation of* $\boldsymbol{\varepsilon}$.

A distributed interpretation assigns a standard DL interpretation to each ontology in the system, as well as an equalizing function that correlates local knowledge into a global domain of interpretation.

Definition 4 (Distributed Interpretation). *Let* $S = \langle \mathbf{O}, \mathbf{A} \rangle$ *be a DS. A* distributed interpretation *of* S *is a pair* $\langle \mathbf{I}, \boldsymbol{\varepsilon} \rangle$ *where* $\mathbf{I}$ *is a family of interpretations indexed by* $\mathbf{O}$, $\boldsymbol{\varepsilon}$ *is an equalizing function for* $\mathbf{I}$, *such that for all* $i \in \mathbf{O}$, I_i *interprets* i *and* $\varepsilon_i : \Delta^{I_i} \rightarrow \Delta_\varepsilon$.

While local satisfiability is the same as standard DL, correspondence satisfaction involves the equalizing function.

Definition 5 (Satisfaction of a Correspondence). *Let* S *be a DS, and* i, j *two ontologies of* S. *Let* $\mathcal{I} = \langle \mathbf{I}, \boldsymbol{\varepsilon} \rangle$ *be a distributed interpretation. We define satisfaction of a correspondence* c *(denoted by* $\mathcal{I} \models_d c$*) as follows:*

$$\begin{array}{lll}
\mathcal{I} \models_d i{:}C \xleftrightarrow{\sqsubseteq} j{:}D & \textit{iff} & \varepsilon_i(C^{I_i}) \subseteq \varepsilon_j(D^{I_j}) \\
\mathcal{I} \models_d i{:}R \xleftrightarrow{\sqsubseteq} j{:}S & \textit{iff} & \varepsilon_i(R^{I_i}) \subseteq \varepsilon_j(S^{I_j}) \\
\mathcal{I} \models_d i{:}C \xleftrightarrow{\perp} j{:}D & \textit{iff} & \varepsilon_i(C^{I_i}) \cap \varepsilon_j(D^{I_j}) = \emptyset \\
\mathcal{I} \models_d i{:}R \xleftrightarrow{\perp} j{:}S & \textit{iff} & \varepsilon_i(R^{I_i}) \cap \varepsilon_j(S^{I_j}) = \emptyset \\
\mathcal{I} \models_d i{:}a \xleftrightarrow{\in} j{:}C & \textit{iff} & \varepsilon_i(a^{I_i}) \in \varepsilon_j(C^{I_j}) \\
\mathcal{I} \models_d i{:}a \xleftrightarrow{=} j{:}b & \textit{iff} & \varepsilon_i(a^{I_i}) = \varepsilon_j(b^{I_j})
\end{array}$$

Additionally, for all local formulas $i{:}\phi$, $\mathcal{I} \models_d i{:}\phi$ iff $I_i \models \phi$ (*i.e.*, local satisfaction is equivalent to global satisfaction of local formulas). A distributed interpretation $\mathcal{I}$ satisfies an alignment A iff it satisfies all correspondences of A (denoted by $\mathcal{I} \models_d A$) and it satisfies an ontology O_i iff it satisfies all axioms of O_i (denoted by $\mathcal{I} \models_d O_i$). When all ontologies and all alignments are satisfied, the DS is satisfied by the distributed interpretation. In which case we call this interpretation a *model* of the system.

[1] We systematicaly use bold face to denote a mathematical family of elements. So, $\mathbf{O}$ denotes $(O_i)_{i \in I}$ where I is a set of indices.

Definition 6 (Model of a DS). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. A distributed interpretation $\mathcal{I}$ is a* model *of S (denoted by $\mathcal{I} \models_d S$), iff:*

- *for all $O_i \in \mathbf{O}$, $\mathcal{I} \models_d O_i$;*
- *for all $A_{ij} \in \mathbf{A}$, $\mathcal{I} \models_d A_{ij}$.*

The set of all models of a DS is denoted by $\mathrm{Mod}(S)$. A formula α is a consequence of a DS ($S \models_d \alpha$) iff $\forall \mathcal{M} \in \mathrm{Mod}(S), \mathcal{M} \models_d \alpha$. This model-theoretic semantics offers special challenges to the reasoning infrastructure, that we discuss in next section.

3 Reasoning in IDDL with Concept Correspondences

In this section, we investigate a reasoning procedure for checking whether or not $S = \langle \mathbf{O}, \mathbf{A} \rangle$ is consistent, in the case when only concepts are put in correspondences. Role correspondences are considered in the next section.

We can reduce the problem of entailment $S \models_d \alpha$ to deciding (in)consistency of a DS when α is either a local GCI ($i{:}C \sqsubseteq D$), a concept correspondence ($i{:}C \xleftrightarrow{\sqsubseteq} j{:}D$ or $i{:}C \xleftrightarrow{\perp} j{:}D$) or a local ABox assertion ($i : C(a)$). Local entailment reduction is straightforward. However, correspondence entailment like $S \models_d i{:}C \xleftrightarrow{\sqsubseteq} j{:}D$ is equivalent to the inconsistency of $S \cup \{i{:}\{a\} \xleftrightarrow{\sqsubseteq} i{:}C\} \cup \{i{:}\{a\} \xleftrightarrow{\perp} j{:}D\}$, where a is a new individual name added to ontology O_i.

When ontologies are correlated with alignments, new deductions may occur. Indeed, cross-ontology knowledge interacts with local knowledge. Moreover, knowledge from one ontology may influence knowledge from another ontology. Besides, local knowledge would also induce cross-ontology knowledge (*i.e.*, alignments). And finally, deductions can be made with and about the alignments alone.

In fact, the difficulty of reasoning in IDDL resides in determining what knowledge propagates from local domains to global domain, or from global to local domains. For instance, if there is a correspondence which asserts disjointness of two concepts from a local ontology then the semantics of the system imposes disjointness of these two concepts in the local ontology. We will show that, in the restricted case when only concept correspondences are allowed, it suffices to propagate only unsatisfiability and non-emptiness of concepts.

Example 1. Let S be the DS composed of $O_1 = \{D_1 \equiv B_1 \sqcap C_1\}$, $O_2 = \{B_2 \sqsubseteq \top, C_2 \sqsubseteq \top\}$ and alignment $A_{12} = \{1{:}B_1 \xleftrightarrow{\sqsubseteq} 2{:}B_2, 1{:}C_1 \xleftrightarrow{\perp} 2{:}B_2, 1{:}D_1 \xleftrightarrow{\sqsupseteq} 2{:}C_2\}$.

We see that $S \models 1{:}B_1 \xleftrightarrow{\perp} 1{:}C_1$. If $B_1 \sqsubseteq \neg C_1$ is added to O_1 (as a consequence of knowledge propagation from the alignments to the ontology) then D_1 becomes unsatisfiable in O_1. From the correspondence $1 : D_1 \xleftrightarrow{\sqsupseteq} 2 : C_2$, it follows that C_2 is unsatisfiable in O_2 as well.

Ex. 1 shows that reasoning on IDDL systems is not trivial and the existing algorithms for reasoning on DL-based ontologies (*e.g.*, tableau algorithms) cannot be directly used.

The principle behind the algorithm is based on the fact that correspondences are similar to axioms, and alignments resemble ontologies. In fact, an alignment represents an ontology which would be interpreted in the global domain of interpretation (see Def. 4). In this algorithm, the alignments will be translated into an ontology (the global ontology). However, this is not enough to check global consistency because local knowledge influences global reasoning. So, the idea consists in extending the global ontology together with the local ontologies by adding specific axioms which represent knowledge propagated through the distributed system.

As a matter of fact, if correspondences are restricted to cross-ontology concept subsumption or disjointness, only concept unsatisfiability and concept non-emptiness[2] can be propogated. Indeed, if a concept is locally interpreted as empty, then its image via the equalizing function is empty too. Conversely, a non-empty set has a non-empty image through ε.

Unfortunately, it is not possible to propagate knowledge by analysing ontologies one by one. A subtle combination of several ontologies **and** alignments can impose unsatisfiability of a locally satisfiable concept (see Ex. 1).

In order to be certain that all concept unsatisfiability and non-emptiness are propagated, our algorithm exhaustively tests each combination of concept unsatisfiability and non-emptiness by explicitely adding these facts, and propagating them accordingly.

In the sequel, we introduce the construction of extended ontologies from $\mathbf{A}$ and $\mathbf{O}$ and we show that the consistency of an IDDL system S is equivalent to the existence of such extended ontologies such that they are consistent.

3.1 Configurations and Extended Ontologies

This section provides the formal definitions which will finally lead to the construction of the extended ontologies mentioned above. A configuration determines whether certain well-chosen concepts in a vocabulary are unsatisfiable or non-empty. In our specific case, the vocabulary in question is defined by the correspondences. It will be proven that it is sufficient to consider concepts appearing in correspondences when dealing with knowledge propagation in IDDL.

More precisely, concepts occurring as the left or right side of correspondences in alignments constitute the vocabulary of an alignment ontology, namely *global vocabulary*. It consists, in turn, of *local vocabularies* which are originated from local ontologies. The following definitions introduce formally the construction of these elements.

Definition 7 (Local Vocabulary). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. We denote by $\mathscr{C}_i$ the set that includes the top concept $\top$ and all (primitive or complex) concepts that appear in the left side of correspondences in A_{ij} or in the right side of correspondences in A_{ji}.*

Definition 8 (Global Vocabulary). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. The set of global concept names of S is $\mathscr{C} = \bigcup_{i \in \mathbf{O}} \{i{:}C \mid C \in \mathscr{C}_i\} \cup \{\top\}$. When $w \subseteq \mathscr{C}_i$, we denote by $\widehat{w}$ the set $\{i{:}C \mid C \in w\}$ of (global) concept names. When $W \subseteq \mathscr{C}$, we denote by $W|_i$ the set $\{C \in \mathscr{C}_i \mid i{:}C \in W\}$.*

[2] In this paper, "concept non-emptiness" means that an interpretation satisfies the system only if the concept is interpreted as a non-empty set.

Example 2. Considering the system of Ex. 1, the local vocabulary $\mathscr{C}_1$ is $\{B_1, C_1, D_1, \top_1\}$ and $\mathscr{C}_2$ is $\{B_2, C_2, \top_2\}$, while the global vocabulary $\mathscr{C}$ is $\{1{:}B_1, 1{:}C_1, 1{:}D_1, 1{:}\top_1, 2{:}B_2, 2{:}C_2, 2{:}\top_2, \top\}$.

As mentioned before Ex. 1 we need to determine the unsatisfiability or non-emptiness of certain concepts but not only the concepts of the vocabularies. It is necessary to know also the unsatisfiability or non-emptiness of all *atomic decompositions* [9] on concepts in a vocabulary. The reason is that, for instance, the non-emptiness of two concepts $C, D \in \mathscr{C}$ does not mean the non-emptiness of $C \sqcap D$ which should be propagated to local ontologies. Concepts defined in Def. 9 express just all atomic decompositions on concepts in a set T.

Definition 9. *Let T be a set of concepts (primitive or complex) including $\top$. For each non empty subset $W \subseteq T$, we define the concept* $C_W^T := (\bigsqcap_{X \in W} X \sqcap \bigsqcap_{X' \in T \setminus W} \neg X')$.

From Def. 9, it follows that all concepts C_W^T are disjoint and their union is equivalent to $\top$. As a consequence, an interpretation of vocabulary T associates to the set of concepts C_W^T a partition of the interpretation domain.

Relying on concepts C_W^T we can define an equivalence relation over the set of interpretations of T as follows: two interpretations belong to an equivalence class if for each subset $W \subseteq T$, they both interpret concept C_W^T as empty, or both interpret it as non empty. The notion of *configuration* defined below represents such equivalence classes. For more convenience, a configuration only indicates the subset of atomic decompositions which will be considered as non-empty, while the others are considered unsatisfiable.

Consequently, a configuration is just a choice of a subset of all atomic decompositions.

Definition 10 (Global Configuration). *Let S be a DS with a set of global concept names $\mathscr{C}$. A* global configuration *of S is a subset Ω of $2^{\mathscr{C}}$.*

Configurations are essential because, as we will show, consistency of a DS can be equated to finding a relevant configuration instead of considering the possibly infinite set of all equalizing functions. However, to achieve this, we must translate the configuration into axioms and assertions which express non-emptiness and unsatisfiability, respectively.

We have prepared necessary elements for constructing the so-called *alignment ontology*. This ontology "axiomatizes" the alignments, which represent inter-ontology knowledge. Apart from axioms expressing correspondences in alignments, an alignment ontology includes additional axioms or assertions representing the global configuration.

Definition 11 (Alignment Ontology). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. Let Ω be a global configuration of S. The* alignment ontology *w.r.t. Ω is an ontology $\widehat{\mathbf{A}}_\Omega$ defined as follows:*

1. *for each $i, j \in \mathbf{O}$, if $i : C \xleftrightarrow{\sqsubseteq} j : D$ (resp. $i : C \xleftrightarrow{\perp} j : D$) is a concept correspondence in $\mathbf{A}$ then $i{:}C \sqsubseteq j{:}D$ (resp. $i{:}C \sqsubseteq \neg j{:}D$) is an axiom of $\widehat{\mathbf{A}}_\Omega$;*

2. *for each* $W \in \Omega$, $C_W \equiv \{a_W\}$ *is an axiom of* $\widehat{\mathbf{A}}_\Omega$ *where* a_W *is a new individual name;*
3. *for each* $W \notin \Omega$, $C_W \sqsubseteq \bot$ *is an axiom of* $\widehat{\mathbf{A}}_\Omega$.

Axiomatization of the alignments renders explicit the constraints imposed by correspondences. The additional axioms or assertions constrain interpretations to belong to the equivalence class represented by configuration Ω.

Example 3. Reconsidering Ex. 1, if we pick, for instance, the configuration $\Omega = 2^{\mathscr{C}} \setminus \{\emptyset\}$ to build an alignment ontology $\widehat{\mathbf{A}}_\Omega$ according to Def. 11, then $\widehat{\mathbf{A}}_\Omega$ is inconsistent because $W = \{1{:}B_1, 1{:}C_1\} \in \Omega$, $\widehat{\mathbf{A}}_\Omega \models (1{:}B_1 \sqcap 1{:}C_1)(a_W)$ and $\widehat{\mathbf{A}}_\Omega \models 1{:}B_1 \sqsubseteq \neg 1{:}C_1$.

The construction of local configurations is very similar to that of global configuration except that compatibility of local configurations with a given global configuration must be taken into account. This compatibility results from the semantics of IDDL system, which imposes that if the image of a set under an equalizing function is not empty then that set must be not empty.

Definition 12 (Local Configuration). *Let* $S = \langle \mathbf{O}, \mathbf{A} \rangle$ *be a DS. Let* Ω *be a global configuration of* S. *For each* $O_i \in \mathbf{O}$, *we define a* local configuration *of* O_i *w.r.t.* Ω *as a subset* Ω_i *of* $2^{\mathscr{C}_i}$. *Moreover, if* $w \in \Omega_i$ *then there must exist* $W \in \Omega$ *such that* $\widehat{w} \subseteq W$.

As discussed at the beginning of this section, knowledge propagation from alignments to local ontologies is crucial to the construction of extended ontologies which preserve the consistency of an IDDL system. A global configuration Ω and a local configuration Ω_i which is compatible with Ω provide necessary elements to define such extended ontologies. The following definition describes how to propagate knowledge from alignments to local ontologies through the determined configurations.

Definition 13 (Extended Ontologies). *Let* $S = \langle \mathbf{O}, \mathbf{A} \rangle$ *be a DS. Let* Ω *be a global configuration of* S. *For each* $O_i \in \mathbf{O}$, *let* Ω_i *be a local configuration w.r.t.* Ω. *The* extended ontology $\widehat{O}_{\Omega_i}$ *w.r.t.* Ω_i *and* Ω *is defined as follows:*

1. $O_i \subseteq \widehat{O}_{\Omega_i}$;
2. *for each* $w \in \Omega_i$, $C_w^{\mathscr{C}_i}(b_w)$ *is an axiom of* $\widehat{O}_{\Omega_i}$, *where* b_w *is a new individual name;*
3. *for each* $w \notin \Omega_i$, $C_w^{\mathscr{C}_i} \sqsubseteq \bot$ *is an axiom of* $\widehat{O}_{\Omega_i}$;
4. *for each* $W \in \Omega$ *and for each* $X \in W|_i$, *we define a new concept* C_W^X *for ontology* $\widehat{O}_{\Omega_i}$ *such that:*
   (a) $C_W^X \sqsubseteq X \sqcap \bigsqcap_{X' \in \mathscr{C}_i \setminus W|_i} \neg X'$ *is an axiom of* $\widehat{O}_{\Omega_i}$;
   (b) $C_W^X(b_W^X)$ *is an axiom of* $\widehat{O}_{\Omega_i}$ *with* b_W^X *a new individual name in* $\widehat{O}_{\Omega_i}$;
   (c) $C_W^X \sqsubseteq C_W^\top$ *is an axiom of* $\widehat{O}_{\Omega_i}$;
5. *for each* $W, W' \subseteq \mathscr{C}$ *such that* $W \neq W'$, $C_W^\top \sqsubseteq \neg C_{W'}^\top$ *is an axiom of* $\widehat{O}_{\Omega_i}$.

Notice that the propagation of knowledge through a global configuration is not straightforward. The non-emptiness expressed by the assertion $C_W \equiv \{a_W\}$ indicates that each concept of W coming from the local vocabulary $\mathscr{C}_i$ must be individually non empty, but not necessarily *conjonctly* non-empty. Consequently, the decomposition of the concept C_W for the propagation as described in the item 4a in Def. 13 is necessary.

The following theorem establishes the most important result in the present section. It asserts that an IDDL system can be translated into an alignment ontology and extended ontologies that preserve the semantics of the IDDL system.

Theorem 1 (DS Consistency). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. S is consistent iff there exist a global configuration Ω of S and a local configuration Ω_i for each $O_i \in \mathbf{O}$ w.r.t. Ω such that the alignment ontology $\widehat{\mathbf{A}}_\Omega$ and the extended local ontologies $\{\widehat{O}_{\Omega_i}\}$ as defined in Def. 11 and Def. 13 are consistent.*

A proof of this theorem is given in a technical report [11].

4 Reasoning with Cross-Ontology Role Subsumption

In this section, we devise a new reasoning procedure which now takes into account cross-ontology role subsumption. The principle behind this improved reasoning task is the same as before, except that configurations must be extended to take into account the roles involved in correspondences. Since most of the definitions necessary for this part are the same or similar as the ones for the previous part, we simply update existing definitions or add new definitions when necessary.

First, a new notion of role vocabulary must be defined, locally or globally.

Definition 14 (Local Role Vocabulary). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. We denote by $\mathscr{R}_i$ the set that includes primitive or complex roles that appear in the left side of correspondences in A_{ij} or in the right side of correspondences in A_{ji} together with their inverse roles (i.e., $R \in \mathscr{R}_i \longleftrightarrow R^- \in \mathscr{R}_i$).*

Definition 15 (Global Role Vocabulary). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. The set of global role names of S is $\mathscr{R} = \bigcup_{i \in \mathbf{O}} \{i{:}R \mid R \in \mathscr{R}_i\}$.*

Now we must define a new kind of configuration which has to be considered in addition to the already defined global and local configurations. However, the treatment of roles is quite different from the treatment of concepts only, because there are interactions between roles and concepts. Therefore, we need to keep track of role satisfiability in addition to concept satisfiability.

We do that by considering a given (concept) configuration Ω which represents a partition of the domain of interpretation. Then, according to this configuration, we define the role configuration as a family of relations over Ω indexed by the set of roles. In other terms, we determine in a role configuration whether there exists a relation R between two sets in the partition Ω.

Definition 16 (Role Configuration). *Let S be a DS with a set of global role names $\mathscr{R}$. Let Ω be a global configuration of S. A* role configuration *of S w.r.t. Ω is a subset Φ_Ω of $\Omega \times \Omega \times \mathscr{R}$.*

The introduction of role configuration leads to additional constraints on the alignment ontology that we summarize in this addendum to Def. 11.

Definition 17 (Alignment Ontology (Revised)). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. Let Ω be a global configuration of S, and let Φ_Ω be a role configuration w.r.t. Ω. The* alignment ontology *w.r.t. Ω and Φ_Ω is an ontology $\widehat{\mathbf{A}}_\Omega$ defined as follows:*

1. *2. and 3. See Def. 11;*
4. *for each $i, j \in \mathbf{O}$, if $i{:}R \xleftrightarrow{\sqsubseteq} j{:}S$ is a role correspondence in $\mathbf{A}$ then $i{:}R \sqsubseteq j{:}S$ is a sub-role axiom of $\widehat{\mathbf{A}}_\Omega$;*
5. *for each $\langle W, W', R \rangle \in \Phi_\Omega$, $C_W \sqsubseteq \exists R.C_{W'}$ is an axiom of $\widehat{\mathbf{A}}_\Omega$;*
6. *for each $\langle W, W', R \rangle \notin \Phi_\Omega$, $C_W \sqsubseteq \forall R.\neg C_{W'}$ is an axiom of $\widehat{\mathbf{A}}_\Omega$;*

The axioms introduced by items 5 and 6 in Def. 17 express the semantics of a role configuration: they impose a R connection or no R connection between two atomic decompositions on concepts in $\mathscr{C}$.

Similarly to local configurations, local role configurations have to satisfy constraints which reflect the propagation of knowledge from the alignment ontology to extended ontologies.

Definition 18 (Local Role Configuration). *Let S be a DS with a set of global role names $\mathscr{R}$. Let Ω be a global configuration of S, let Φ_Ω be a role configuration w.r.t. Ω, and Ω_i a local configuration of O_i w.r.t. Ω. A* local role configuration *of O_i w.r.t. Ω, Ω_i and Φ_Ω is a subset Φ_{Ω_i} of $\Omega_i \times \Omega_i \times \mathscr{R}_i$ such that $\langle w, w', R \rangle \in \Phi_{\Omega_i}$ implies that there exists $W, W' \in \Omega$ such that $w \subseteq W|_i$, $w' \subseteq W'|_i$ and $\langle W, W', i{:}R \rangle \in \Phi_\Omega$.*

The extended ontologies are now further extended with axioms which involve roles.

Definition 19 (Extended Ontologies (Revised)). *Let $S = \langle \mathbf{O}, \mathbf{A} \rangle$ be a DS. Let Ω be a global configuration and Φ_Ω a role configuration of S. For each $O_i \in \mathbf{O}$, let Ω_i be a local configuration w.r.t. Ω and Φ_{Ω_i} be a local role configuration w.r.t. Ω, Ω_i and Φ_Ω. The* extended ontology *$\widehat{O}_{\Omega_i}$ w.r.t. Ω_i, Ω, Φ_Ω and Φ_{Ω_i} is defined as follows:*

1. *2. 3. 4. and 5. See Def. 13;*
6. *for each $\langle w, w', R \rangle \in \Phi_{\Omega_i}$, $(C_w^{\mathscr{C}_i} \sqcap \exists R.C_{w'}^{\mathscr{C}_i})(b^R_{w,w'})$ is an axiom of $\widehat{O}_{\Omega_i}$, where $b^R_{w,w'}$ is a new individual name;*
7. *for each $\langle w, w', R \rangle \notin \Phi_{\Omega_i}$, $C_w^{\mathscr{C}_i} \sqsubseteq \forall R.\neg C_{w'}^{\mathscr{C}_i}$ is an axiom of $\widehat{O}_{\Omega_i}$;*
8. *for each $W, W' \subseteq \Omega$, and each $R \in \mathscr{R}_i$, we define a new concept name $C^R_{W,W'}$ for ontology $\widehat{O}_{\Omega_i}$ such that:*
   (a) $C^R_{W,W'} \sqsubseteq C^\top_W$ is an axiom of $\widehat{O}_{\Omega_i}$;
   (b) if $\langle W, W', i : R \rangle \in \Phi_\Omega$ then $C^R_{W,W'} \sqsubseteq \exists R.C^{R^-}_{W',W}$ and $C^R_{W,W'}(\beta^R_{W,W'})$ are axioms of $\widehat{O}_{\Omega_i}$ with $\beta^R_{W,W'}$ a new individual name;
   (c) else, $C^R_{W,W'} \sqsubseteq \forall R.\neg C^{R^-}_{W',W}$ is an axiom of $\widehat{O}_{\Omega_i}$;
9. *for each $R \in \mathscr{R}_i$, $\exists R.\top \sqsubseteq \bigsqcup_{W,W' \in \Omega} C^R_{W,W'}$ is an axiom of $\widehat{O}_{\Omega_i}$.*

In the previous definition, item 6 means that a triple $\langle w, w', R\rangle$ in the local role configuration determines the existence of a relation R between some member of $C_w^{\mathscr{C}_i}$ and some member of $C_{w'}^{\mathscr{C}_i}$. Conversely, item 7 means that whenever a triple $\langle w, w', R\rangle$ is not in the local role configuration, then concepts $C_w^{\mathscr{C}_i}$ and $C_{w'}^{\mathscr{C}_i}$ are not related through R. Item 8 adds a concept $C_{W,W'}^R$ which represents the set of elements of local concept $C_W^\top$ which have their counterparts in global concept C_W and are in relation through R with elements which have their own counterparts in $C_{W'}$. Finally, item 9 asserts that any element involved in a relation R must belong to one of the newly introduced sets $C_{W,W'}^R$ for some W and W'. This last item is important to ensure that the role structure is correctly propagated.

Theorem 2 (DS Consistency). *Let $S = \langle \mathbf{O}, \mathbf{A}\rangle$ be a DS. S is consistent iff there exist a global configuration Ω of S, a role configuration Φ_Ω w.r.t. Ω, local configurations Ω_i for all $O_i \in \mathbf{O}$ w.r.t. Ω and local role configurations Φ_{Ω_i} w.r.t. Ω, Ω_i and Φ_Ω, such that the alignment ontology $\widehat{\mathbf{A}}_\Omega$ and the extended local ontologies $\{\widehat{O}_{\Omega_i}\}$ as defined in Def. 17 and Def. 19 are consistent.*

5 Algorithms and Improvements

In this section, we try to devise an explicit algorithm for checking the consistency of a distributed system in IDDL. We first present a naive algorithm, as a direct application of Theo. 1. Then, we propose a simple optimization for this particular problem. Finally, we show how the very same principle can be used in a less expressive setting to ensure a tractable consistency checking procedure.

5.1 Naive Algorithm

Theo. 1 provides enough information for building a naive but correct and complete algorithm, which corresponds to an exhaustive traversal of all possible configurations (see Algo. 1).

```
input : $S = \langle \mathbf{O}, \mathbf{A}\rangle$ with $\mathbf{A} = \{A_{ij} \mid i, j \in \mathbf{O}\}$
output: IsConsistent($S$)
1 foreach global configuration $\Omega \subseteq 2^{\mathscr{C}}$ do
2     if Consistent($\widehat{\mathbf{A}}_\Omega$) then
3         foreach family of local configurations $(\Omega^i)_{i\in\mathbf{O}}$ do
4             if Consistent($\widehat{O}_{\Omega^i}$) for all $i \in \mathbf{O}$ then
5                 return true;
6 return false;
```

Algorithm 1. Consistency($\langle \mathbf{O}, \mathbf{A}\rangle$)

Property 1. Given c the number of global concepts in $\mathscr{C}$ and N the number of ontologies in the system, the number of calls for consistency checking of extended ontologies in Alg. 1 is bounded by $N2^{(2^{c+1})}$. Moreover, the size of the extended ontologies to be checked is in the order of $O(2^c)$.

Proof. There are as many global configurations as there are subsets of $2^{\mathscr{C}}$, *i.e.*, $2^{(2^c)}$. For each global configuration, all local configurations have to be tested, for all ontologies. The number of local configuration, for a given ontology in the system, is bounded by $2^{(2^c)}$, and there are N ontologies to be checked. So the total number of consistency checking is bounded by $N2^{(2^c)} \cdot 2^{(2^c)} = N2^{(2^{c+1})}$.

Prop. 1 shows that the complexity of Alg. 1 can be determined from that of consistency checking of extended ontologies. Whatever the local algorithm complexity, the complexity of this global consistency checking algorithm is at least in 2EXPTIME. In the case when the local algorithm is itself in 2EXPTIME (which can happen with some tableau algorithm over very expressive description logics), the global consistency checking algorithm is in 3EXPTIME.

This high intractibility must be balanced with the good properties that it guarantees. First, the algorithm proves that our distributed formalism is decidable whenever local logics are decidable. Second, the actual local logics need not be known, as well as the local decision procedure. Therefore, ontologies can be encapsulated in an interface which only communicate the consistency of its internal ontology extended with well defined axioms. Moreover, the expressiveness of the axioms added to the extended ontologies are restricted to $\mathcal{ALC}$, which is simple enough to cover many existing DL reasoners. The goal of the next section is to explore possible optimization for this particular problem.

5.2 Optimizing the Algorithm

This section proposes several simple optimizations which significantly decrease complexity. Unfortunately, they do not change the class of complexity, but help approaching tractability in "favorable situations". The goal is to reduce as much as possible the number of configurations that must be considered. To simplify this discussion, we only focus on the global concept configurations. We consider three complementary methods for improving the algorithm.

Using Correspondences. This first method takes advantage of the correspondences to systematically decrease the possible configurations. Indeed, it may be noticed that if $i:C \overset{\sqsubseteq}{\leftrightarrow} j:D \in A_{ij}$, then the non-emptiness of C implies the non-emptiness of D. Therefore, it is not necessary to inspect configurations containing $W \subseteq \mathscr{C}$ such that $C \in W$ and $D \notin W$. Additionally, if $i:C \overset{\perp}{\leftrightarrow} j:D \in A_{ij}$, then it is not necessary to inspect configurations containing $W \subseteq \mathscr{C}$ such that $C, D \in W$ since $i:C \sqcap D$ is necessarily empty. This can decrease the search space a lot. For instance, if there are n concepts $C_1, \ldots, C_n$ such that $i:C_k \overset{\sqsubseteq}{\leftrightarrow} j:C_{k+1}$, then the non-emptiness of any C_k

implies the non-emptiness of all C_l for all $l \geq k$. So, in the best possible situation, the number of configurations to be tested would be linear in the size of the alignements. Nonetheless, in the worst situation, the number of configurations is still exponential. Indeed, if there are n concepts $D_1, \ldots, D_n$ and a concept C such that $i{:}C \overset{\sqsubseteq}{\leftrightarrow} j{:}D_k$ for all k, then the non-emptiness of C implies the non-emptiness of all D_k, but when C is empty, the concepts D_k can be independently empty or non-empty. However interesting compared to the brute force approach, this optimization still places the algorithm in the double exponential class. We admit that it is still too high, but this worst case complexity can be avoided in many cases using the two following optimizations.

Using Backtracking Techniques. Thanks to Theo. 1, the problem of checking consistency has been reformulated into finding a configuration. We can notice that an appropriate configuration can be determined step by step with a decision tree, by deciding whether a given subset $W \subseteq \mathscr{C}$ is in the configuration or not. In fact, each node of the tree asks whether C_W is empty or not. In case C_W can neither be empty nor non-empty, the algorithm must backtrack and try another decision for the previous subset considered. Fig. 1 shows a part of a possible decision tree.

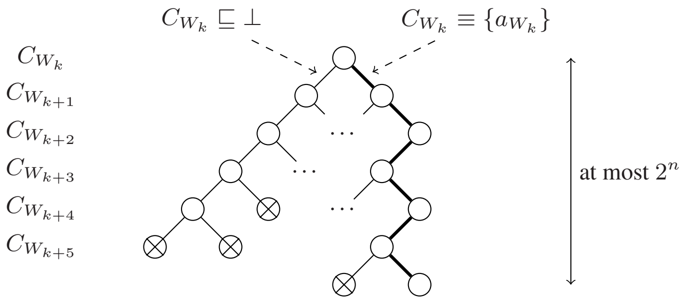


Fig. 1. At each node, the left branch indicates that the concept C_W is asserted as an empty concept ($C_W \sqsubseteq \bot$), while the right branch indicates a non empty concept ($C_W \equiv \{a_W\}$ for a new a_W). The thick path indicates a possible configuration for the distributed system.

Other backtracking techniques like backjumping may be used. With such a method, the number of calls to local reasoners may be reduced to 2^n in favorable cases, not to mention additional reductions due to the previous optimization.

Additional Optimization. In the course of reasoning at a certain level of the decision tree, if it can be proved that a concept C is empty ($C \sqsubseteq \bot$), then it can also be asserted that all conjunctions of C with any other concepts are also empty. More precisely, for all $W \subseteq \mathscr{C}$ such that $C \in W$, the system implies that $C_W \sqsubseteq \bot$, so all configurations containing W can be eliminated. This further decreases the search space. Consequently, we conjecture that there are practical cases where reasoning with our algorithm can be carried out, in spite of the very high worst case complexity. Nonetheless, since these optimizations are still insufficient to treat hard cases practically, we study another possible improvement that can be done when the alignments are less expressive.

5.3 Reducing the Expressivity of Alignments

The highly intractable complexity of the algorithm for checking consistency of an IDDL system is originated from the fact that it may propagate to local ontologies a double exponential number of configurations with exponential size. These configurations represent the structure of models for alignment ontologies which are expressed in a sublogic of $\mathcal{ALC}$. We can simplify the structure of models by removing cross-ontology concept and role disjointness from the alignment language. Tableau-based algorithms, for example in [1], can generate a singleton model for a consistent ontology involving only subsumption axioms between primitive concepts or roles, *i.e.*, each configuration represents now the structure of a singleton model. Therefore, all concepts are interpreted either as the singleton or the empty set. Consequently, it is sufficient to represent models by a set of non-empty primitive concepts.

If $\mathscr{C}$ denotes the global vocabury of an IDDL system (Def. 8) then a global configuration is now defined as $\Omega \subseteq \mathscr{C}$ and the algorithm needs to call to local reasoners at most $2^{\mathscr{C}}$ times with these polynomial configurations. This is true because, in absence of disjointness, testing whether $\bigsqcap_{X \in W} X \sqcap \bigsqcap_{X' \in T \setminus W} \neg X'$ is empty or not can be reduced to testing whether each primitive concept X are empty or equal to an identified singleton.

6 Conclusion and Future Work

We have proposed a reasoning procedure for IDDL which determines the consistency of a distributed system of DL ontologies and alignments. On the one hand, it only requires the minimal support of $\mathcal{ALC}$ reasoning locally, while there is no upper bound expressivity. On the other hand, alignments are currently limited to cross-ontology concept subsumption or disjointness and role subsumption. This restriction on the expressiveness of alignment language is not really severe since alignments produced by almost all ontology matching algorithms [5] are expressible within this restricted alignment language. Furthermore, the majority of these algorithms (OLA, AROMA, Falcon-AO, *etc*. [5]) yields only cross-ontology concept or role subsumption. This meets exactly the reduction of expressiveness of alignment language presented in Sect.5.2.

Several research directions are considered to continue this work. We plan to further optimize the algorithm. This will lead to a distributed implementation taking advantage of various reasoners encapsulating ontologies of unknown complexity. In particular, this would fit quite well in our modular framework presented in [6]. There are also potential optimization when local expressivity is limited to a logic known in advance. Another direction would involve peer reasoning, which means defining the inferences produced by a local reasoner taking advantage of global knowledge in a network of aligned ontologies.

Finally, our goal is to revise the consistency checking procedure by taking into account role disjointness. Eventually, we hope to extend the expressivity of the alignment language, by adding specific constructors in line with the expressive ontology mapping language proposed in [4].

References

1. Baader, F.: Tableau Algorithms for Description Logics. In: Dyckhoff, R. (ed.) TABLEAUX 2000. LNCS, vol. 1847, pp. 1–18. Springer, Heidelberg (2000)
2. Bao, J., Caragea, D., Honavar, V.: On the Semantics of Linking and Importing in Modular Ontologies. 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. 72–86. Springer, Heidelberg (2006)
3. Borgida, A., Serafini, L.: Distributed Description Logics: Assimilating Information from Peer Sources. Journal on Data Semantics I 2008, 153–184 (2003)
4. Euzenat, J., Scharffe, F., Zimmermann, A.: Expressive alignment language and implementation. Deliverable 2.2.10, Knowledge Web NoE (2007), ftp://ftp.inrialpes.fr/pub/exmo/reports/kweb-2210.pdf
5. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
6. Euzenat, J., Zimmermann, A., Freitas, F.: Alignment-based modules for encapsulating ontologies. In: Cuenca-Grau, B., Honavar, V.G., Schlicht, A., Wolter, F. (eds.) Second International Workshop on Modular Ontologies (WOMO 2007), pp. 32–45 (2007)
7. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: $\mathcal{E}$-connections of abstract description systems. Artificial Intelligence 156(1), 1–73 (2004)
8. Lenzerini, M.: Data integration: a theoretical perspective. In: Popa, L. (ed.) Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Madison, Wisconsin, USA, June 3-5, 2002, pp. 233–246. ACM Press, New York (2002)
9. Ohlbach, H.J., Koehler, J.: Modal Logics, Description Logics and Arithmetic Reasoning. Artificial Intelligence 109(1–2), 1–31 (1999)
10. Zimmermann, A.: Integrated Distributed Description Logics. In: Calvanese, D., Franconi, E., Haarslev, V., Lembo, D., Motik, B., Tessaris, S., Turhan, A.-Y. (eds.) Proceedings of the 20th International Workshop on Description Logics DL 2007, pp. 507–514. Bolzano University Press (2007)
11. Zimmermann, A., Le Duc, C.: Reasoning on a Network of Aligned Ontologies. Technical report, INRIA (March 2008), http://hal.inria.fr/inria-00267808/fr/
12. Zimmermann, A., Euzenat, J.: Three Semantics for Distributed Systems and their Relations with Alignment Composition. 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. 16–29. Springer, Heidelberg (2006)

Lexicographical Inference over Inconsistent DL-Based Ontologies

Jianfeng Du[1,2], Guilin Qi[3], and Yi-Dong Shen[1]

[1] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences
[2] Graduate University of the Chinese Academy of Sciences
Beijing 100080, China
jfdu,ydshen@ios.ac.cn
[3] AIFB, Universität Karlsruhe, D-76128 Karlsruhe, Germany
gqi@aifb.uni-karlsruhe.de

Abstract. Logical inconsistency may often occur throughout the development stage of a DL-based ontology. We apply the *lexicographic inference* to reason over inconsistent DL-based ontologies without repairing them first. We address the problem of checking consequences in a $\mathcal{SHIQ}$ ontology that are classically inferred from every consistent (or coherent) subontology having the highest *lexicographic precedence*. We propose a method for compiling a $\mathcal{SHIQ}$ ontology to a propositional program so that the problem can be solved in polynomial calls to a SAT solver. We prove that this time complexity is worst-case optimal in data complexity. In order to make the method more scalable, we also present partition-based techniques to optimize the calling of SAT solvers.

1 Introduction

Ontologies play a core role for the success of the Semantic Web (SW) as they provide shared vocabularies for different domains. The Web Ontology Language (OWL) [24] is a standard language for modeling ontologies in the SW, which is based on Description Logics (DLs) [1]. The quality of ontologies is highly important for the SW technology. However, in practice it is difficult to construct an error free or logically consistent DL-based ontology. Logical inconsistency may often occur in different scenarios, such as ontology modeling, evolution, migration and merging [11,29]. For example, if ontologies such as SUMO and CYC are directly merged into a single ontology, there will be misalignments of concepts that introduce logical inconsistency [28].

Given an inconsistent ontology, one may want to repair it so as to apply standard reasoners to access its (implicit) information. To fulfill this requirement, some methods (e.g. [7,15,23,28,29]) emerge. They repair inconsistent ontologies through debugging or diagnosing. Nevertheless, as pointed out by Haase *et al.* [11], in some cases consistency cannot be guaranteed at all and inconsistency cannot be repaired, still one wants to reason over ontologies in order to support information access and integration of new information. Hence, some other methods (e.g. [13,19,18]) emerge to fulfill the latter requirement. They tolerate inconsistency and apply non-standard reasoning methods to obtain meaningful answers from an inconsistent ontology. Our method given in this paper belongs to the latter family of methods.

The notion of ordering plays a crucial role in handling inconsistency as it gives clues to tell which information is more important and should be kept. The well-known *lexicographic ordering* is defined for *stratified knowledge bases* in propositional logic [3], where a knowledge base, viewed as a set of formulas, is divided into a set of strata with priorities. A subbase is lexicographically preferable to another one if it contains more formulas in strata with higher priorities. A *lex-maximal consistent subbase* is defined as a subbase that has the highest lexicographic precedence. To reason with inconsistency, the *lexicographic inference* based on such ordering checks consequences that are classically inferred from every lex-maximal consistent subbase. Since many advantages of lexicographic inference have been shown in the literature [3,4], such as the flexibility for utilizing priority information (e.g., priorities of different sources in the situation of ontology merging) and the conformity to the minimal-change point of view, we apply lexicographic inference to DLs.

The major challenge in applying lexicographic inference to DLs lies on the practicality of the computational aspect, because both reasoning in expressive DLs and lexicographic inference in propositional logic are already computationally hard. To take this challenge, we develop a method that is expected to work well on $\mathcal{SHIQ}$ [12] ontologies with simple terminologies and large ABoxes. The method checks *lex-consistent consequences* (resp. *lex-coherent consequences*) of a $\mathcal{SHIQ}$ ontology that are classically inferred from every *lex-maximal consistent* (resp. *coherent*[1]) *subontology*. Basically, the method first compiles the input $\mathcal{SHIQ}$ ontology into a propositional program, then performs the checking over the propositional program by polynomial calls to a SAT solver. We prove that this time complexity is worst-case optimal in *data complexity*, i.e. the complexity measured in the size of the ABox only. The compilation is based on an extension of the KAON2 transformation [14,20] in which *decision atoms* are embedded, where decision atoms are new nullary atoms one-to-one corresponding to axioms in the original ontology. In order to make the method more scalable, we further adapt the partitioning technique in [7] to decompose the compiled propositional program, and develop a novel algorithm for using the partitioning results to check lex-consistent consequences. For the problem of checking a lex-coherent consequence, we first reduce it to the problem of checking a lex-consistent consequence, then solve it in the same way.

By now we have not fully tested the proposed method but a complete implementation is under way. Some relevant experimental results were reported in [7]. The method given in [7][2] similarly applies SAT solvers to compute consistent subontologies with certain maximality, such as those ones having the maximum number of ABox axioms[3]. It was shown [7] that for a $\mathcal{SHIQ}$ ontology, even though the ABox is large (i.e. has over tens of thousands of axioms), as long as the KAON2 transformation can reduce the terminology to over hundreds of DATALOG$^{\vee}$ [9] rules within which only a few are disjunctive or with equality, the subsequent partitioning step yields small propositional subprograms that can be efficiently handled by SAT solvers. Based on the fact

[1] An ontology is called coherent if all atomic concept in it are satisfiable.

[2] The method employs the original KAON2 transformation and has a restriction that the terminology must be fixed and consistent. Such restriction is removed in the current work.

[3] They are actually lex-maximal consistent subontologies of a stratified ontology $\mathcal{O} = (\mathcal{O}_1, \mathcal{O}_2)$, where $\mathcal{O}_1$, consisting of all terminological axioms, is the stratum with higher priority; and $\mathcal{O}_2$, consisting of all ABox axioms, is the stratum with lower priority.

that the extended KAON2 transformation spends almost all the time on the terminology (it directly translates atomic ABox axioms to ground clauses), the effectiveness of the partitioning step in decomposing a large propositional program into much smaller subprograms, as well as the efficiency of current powerful SAT solvers, we expect that the proposed method works well on $\mathcal{SHIQ}$ ontologies with simple terminologies and large ABoxes.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 gives some background on $\mathcal{SHIQ}$ and lexicographic inference. Section 4 formally defines lex-consistent and lex-coherent consequences for lexicographic inference in DLs. Section 5 presents our method for checking a lex-consistent (or lex-coherent) consequence of a stratified $\mathcal{SHIQ}$ ontology. Section 6 concludes. Due to the space limitation, we do not give (full) proofs of all theorems in this paper but refer the reader to our technical report [6] for more details.

2 Related Work

There exist some computational methods for lexicographic inference in DLs. In the work of Meyer *et al.* [19], the lexicographic inference and its refined version are respectively applied to $\mathcal{ALC}$ and its extension with *cardinality restrictions on concepts*. These inferences are computed through a *disjunctive DL knowledge base* (DKB for short) compiled from the original ontology. A *lex-consistent consequence* of the original ontology amounts to a consequence of the compiled DKB that is classically inferred from all disjuncts of the compiled DKB, where each disjunct is a DL-based ontology. In the work of Qi *et al.* [26], two other refined versions of lexicographic inference are proposed. The corresponding computational methods are also DKB-based. It should be noted that the DKB-based methods have a very high computational complexity. First, the compilation of a DKB needs up to exponential number of DL satisfiability tests wrt the number of axioms in the original ontology. Note that a satisfiability test in $\mathcal{SHIQ}$ is already NP-complete in data complexity [14]. Second, the checking of a consequence of a DKB is performed over all its disjuncts. Since the number of disjuncts can be exponential in the the number of axioms in the original ontology, the checking phase may need another exponential number of DL satisfiability tests. In contrast, our proposed method performs polynomial number of propositional satisfiability tests wrt the number of axioms in the original ontology in both the compiling phase and the checking phase. Each such satisfiability test is also NP-complete in data complexity and can further be optimized by our proposed partition-based techniques.

There exist other methods for reasoning over inconsistent DL-based ontologies [13,17,18,22,27]. As ours, most of them first specify the preferred consistent subontologies, then check consequences classically inferred from those subontologies. The method proposed in [13] first selects a consistent subontology based on a selection function, which is defined on the syntactic or semantic relevance, then reasons over the selected subontology. Such selected subontology is not always maximal, so the inference is less satisfactory from the minimal-change point of view. The methods given in [27] respectively extend *possibilistic* and *linear order* inferences in DLs by exploiting uncertainty degrees on DL axioms. Each extended inference selects a consistent subontology that keeps more DL axioms than the original one does, but the selected subontology is still often not maximal, so these extended inferences are also less satisfactory from the minimal-change point of view. The method given in [17] essentially checks consequences that are classically inferred from every maximal consistent subontology. It does not consider priority information on DL axioms and has a restriction that the terminology must be fixed and consistent. The reasoning methods proposed in [22] and [18] adopt a different idea. To tolerate inconsistency, they weaken an interpretation from two truth values to four truth values. Thus they result in a completely different reasoning mechanism for DL-based ontologies.

3 Preliminaries

The $\mathcal{SHIQ}$ description logic [12] is highly related to OWL DL [24]. It semantically equals OWL DL without nominals and datatype specifications but with qualified number restrictions.

Given a set of role names N_R, a role is either some $R \in N_R$ or an inverse role R^- for $R \in N_R$. An *RBox* $\mathcal{O}_\mathcal{R}$ is a finite set of *transitivity axioms* $Trans(R)$ and *role inclusion axioms* $R \sqsubseteq S$, for R and S roles. For $R \in N_R$, we set $Inv(R) = R^-$ and $Inv(R^-) = R$, and assume that $R \sqsubseteq S \in \mathcal{O}_\mathcal{R}$ implies $Inv(R) \sqsubseteq Inv(S) \in \mathcal{O}_\mathcal{R}$ and $Trans(R) \in \mathcal{O}_\mathcal{R}$ implies $Trans(Inv(R)) \in \mathcal{O}_\mathcal{R}$. A role R is called *transitive* if $Trans(R) \in \mathcal{O}_\mathcal{R}$; *simple* if it has not any transitive subrole. Given a set of concept names N_C, the set of $\mathcal{SHIQ}$ concepts is the minimal set such that each $A \in N_C$ is a $\mathcal{SHIQ}$ concept (called an *atomic concept*) and, for C and D $\mathcal{SHIQ}$ concepts, R a role, S a simple role, and n a positive integer, $\top, \bot, \neg C, C \sqcap D, C \sqcup D, \exists R.C, \forall R.C, \leq n\ S.C$ and $\geq n\ S.C$ are also $\mathcal{SHIQ}$ concepts. A *TBox* $\mathcal{O}_\mathcal{T}$ is a finite set of *concept inclusion axioms* $C \sqsubseteq D$, where C and D are $\mathcal{SHIQ}$ concepts. An *ABox* $\mathcal{O}_\mathcal{A}$ is a set of *concept membership axioms* $C(a)$, *role membership axioms* $R(a, b)$, and *(in)equality axioms* $a \approx b, a \not\approx b$, where C is a $\mathcal{SHIQ}$ concept, R a role, and a and b individuals. The axioms $C(a)$, $R(a, b)$, $a \approx b$ and $a \not\approx b$ are also called *ABox axioms*; called *atomic ABox axioms* if C is a concept name and R is a role name.

A $\mathcal{SHIQ}$ ontology $\mathcal{O}$ consists of an RBox $\mathcal{O}_\mathcal{R}$, a TBox $\mathcal{O}_\mathcal{T}$, and an ABox $\mathcal{O}_\mathcal{A}$. $\mathcal{O}_\mathcal{R} \cup \mathcal{O}_\mathcal{T}$ is also called the *terminology* of $\mathcal{O}$.

As a description logic, $\mathcal{SHIQ}$ inherits its semantics from first-order logic by the standard translations known e.g. from [14]. Let π denote the operator for mapping a $\mathcal{SHIQ}$ ontology to a first-order logic program as given in [14], which can also be found in our technical report [6]. Then, $\mathcal{O}$ is *consistent* (or *satisfiable*) iff there exists a first-order model of $\pi(\mathcal{O})$. A concept C is *satisfiable* in $\mathcal{O}$ iff there exists a first-order model of $\pi(\mathcal{O})$ that satisfies $C(a)$ for some individual a. $\mathcal{O}$ is *coherent* iff all atomic concepts in it are satisfiable.

The lexicographic inference is originally defined for *stratified knowledge bases* in propositional logic [3]. A stratified knowledge base S is viewed as a set of formulas and is divided into a set of strata $\{S_1, \ldots, S_n\}$, where S_i is a subset of formulas in S such that $S = \bigcup_{i=1}^n S_i$ and $S_j \cap S_k = \emptyset$ for any $j \neq k$. S can be written as $(S_1, \ldots, S_n)$. The formulas in S_i have the same priority and have a higher priority than the ones in S_{i+1}. Let $|S|$ denote the number of formulas in the knowledge base S. The *lexicographic ordering* is a complete preordering between any two subbases $A = (A_1, \ldots, A_n)$ and $B = (B_1, \ldots, B_n)$ of $S = (S_1, \ldots, S_n)$, where $A_i \subseteq S_i$ and $B_i \subseteq S_i$ for any i, defined as follows: $A <^{lex} B$ iff there exists i such that $|A_i| < |B_i|$ and $|A_j| = |B_j|$ for any $j < i$; $A =^{lex} B$ iff $|A_i| = |B_i|$ for any i. By $A \leq^{lex} B$ we denote $A <^{lex} B$ or $A =^{lex} B$. S is called *consistent* if it has a model. A *lex-maximal consistent subbase* S' of S is defined as a consistent subbase of S such that for any consistent subbase S'' of S, $S'' \leq^{lex} S'$. A formula ψ is called a *lex-consistent consequence* of S if for any lex-maximal consistent subbase S' of S, $S' \models \psi$, i.e., every model of S' is a model of ψ. Given a stratified knowledge base S and a formula ψ, the *lexicographic inference* problem checks if ψ is a lex-consistent consequence of S.

4 Lexicographic Inference in DLs

In order to apply lexicographic inference to DLs, we view a DL-based ontology as a set of axioms (i.e. as a syntactic object). This syntactic approach to treating DL-based ontologies is commonly used in handling inconsistency [11]. From this point of view, two DL-based ontologies are regarded as the same iff they have the same set of axioms.

A *stratified ontology* $\mathcal{O}$ is an ontology divided into a set of strata $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$, where $\mathcal{O}_i$ is a subset of axioms in $\mathcal{O}$ such that $\mathcal{O} = \bigcup_{i=1}^{n} \mathcal{O}_i$ and $\mathcal{O}_j \cap \mathcal{O}_k = \emptyset$ for any $j \neq k$. $\mathcal{O}$ is written as $(\mathcal{O}_1, \ldots, \mathcal{O}_n)$, where the axioms in $\mathcal{O}_i$ have the same priority and have a higher priority than the ones in $\mathcal{O}_{i+1}$. Let $|\mathcal{O}|$ denote the number of axioms in the ontology $\mathcal{O}$. Then the notions of *lexicographic ordering* and *lex-maximal consistent subontology* are defined analogously as in stratified knowledge bases by treating axioms as formulas [19]. Since an incoherent ontology cannot deduce nontrivial consequences on the unsatisfiable concepts, one may expect that a lex-maximal subontology is not only consistent but also coherent. For example, in the following incoherent but consistent ontology $\mathcal{O} = \{A \sqsubseteq B, A \sqsubseteq \neg B, B(a)\}$, we trivially have $\mathcal{O} \not\models A(x)$ for any individual x because A is unsatisfiable in $\mathcal{O}$. Hence, we introduce the notion of *lex-maximal coherent subontology*. A lex-maximal coherent subontology $\mathcal{O}'$ of $\mathcal{O}$ is defined as a coherent subontology of $\mathcal{O}$ such that for any coherent subontology $\mathcal{O}''$ of $\mathcal{O}$, $\mathcal{O}'' \leq^{lex} \mathcal{O}'$. Two sorts of lexicographic consequences in DLs are defined below.

Definition 1. For a stratified ontology $\mathcal{O}$, an axiom ax is called a *lex-consistent consequence* (resp. *lex-coherent consequence*) of $\mathcal{O}$, written $\mathcal{O} \vdash^{lex}_{cons} ax$ (resp. $\mathcal{O} \vdash^{lex}_{cohe} ax$), if for any lex-maximal consistent (resp. coherent) subontology $\mathcal{O}'$ of $\mathcal{O}$, $\mathcal{O}' \models ax$, i.e., every model of $\mathcal{O}'$ is a model of ax.

It should be noted that a lex-maximal coherent subontology is not necessarily a lex-maximal consistent subontology, and vice versa. There is no straightforward correspondence between lex-consistent consequences and lex-coherent consequences, as shown in the following example.

Example 1. Let $\mathcal{O} = (\{A \sqsubseteq \bot\}, \{A(a)\})$. Then $\mathcal{O}' = (\emptyset, \{A(a)\})$ is the unique lex-maximal coherent subontology of $\mathcal{O}$, but it is not a lex-maximal consistent subontology of $\mathcal{O}$, because $\mathcal{O}'' = (\{A \sqsubseteq \bot\}, \emptyset)$ is consistent and $\mathcal{O}' <^{lex} \mathcal{O}''$. On the other hand, $\mathcal{O}''$ is the unique lex-maximal consistent subontology of $\mathcal{O}$, but it is not coherent. Hence, $\mathcal{O} \vdash^{lex}_{cohe} A(a)$ but $\mathcal{O} \not\vdash^{lex}_{cons} A(a)$; $\mathcal{O} \vdash^{lex}_{cons} A \sqsubseteq \bot$ but $\mathcal{O} \not\vdash^{lex}_{cohe} A \sqsubseteq \bot$.

In this paper we consider checking both sorts of lexicographic consequences, where the axiom ax in Definition 1 can be a concept membership axiom $C(a)$ or a concept inclusion axiom $C \sqsubseteq D$. The following theorem shows a reduction from the problem of checking a lex-coherent consequence to that of checking a lex-consistent consequence. So we focus on the problem of checking a lex-consistent consequence.

Theorem 1 ([6]). *Let $\mathcal{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$ be a stratified DL-based ontology, and ax an axiom. Then $\mathcal{O} \vdash^{lex}_{cohe} ax$ iff $\mathcal{O}' \vdash^{lex}_{cons} ax$, where $\mathcal{O}' = (\mathcal{A}, \mathcal{O}_1, \ldots, \mathcal{O}_n)$ and $\mathcal{A} = \{A(a) \mid A$ is an atomic concept in $\mathcal{O}$, and a is a new globally unique individual not occurring in $\mathcal{O}$ and $ax\}$.*

Proof sketch. It is sufficient to show a bijection between the set of lex-maximal coherent subontologies of $\mathcal{O}$ and the set of lex-maximal consistent subontologies of $\mathcal{O}'$. (1) Let $S = (S_1, \ldots, S_n)$ be a lex-maximal coherent subontology of $\mathcal{O}$. Then $S' = (\mathcal{A}, S_1, \ldots, S_n)$ is obviously consistent. It can be shown that S' is a lex-maximal consistent subontology of $\mathcal{O}'$. (2) Let $S = (S_0, S_1, \ldots, S_n)$ be a lex-maximal consistent subontology of $\mathcal{O}'$. Since $\{\mathcal{A}, \emptyset, \ldots, \emptyset\}$ is consistent, we have $S_0 = \mathcal{A}$, so $S' = (S_1, \ldots, S_n)$ is coherent. It can be shown that S' is a lex-maximal coherent subontology of $\mathcal{O}$. □

5 Computing Lexicographic Inference in $\mathcal{SHIQ}$

The number of lex-maximal consistent subontologies of a stratified $\mathcal{SHIQ}$ ontology $\mathcal{O}$ can be exponential in $|\mathcal{O}|$ (even in $|\mathcal{O}_{\mathcal{A}}|$, the number of ABox axioms in $\mathcal{O}$). Take an ontology $\mathcal{O}^\dagger = (\mathcal{O}^\dagger_1, \ldots, \mathcal{O}^\dagger_n)$ for example, where $\mathcal{O}^\dagger_1 = \{A \sqcap B \sqsubseteq \bot\}$ and $\mathcal{O}^\dagger_i = \{A(a_i), B(a_i)\}$ for $2 \leq i \leq n$. $\mathcal{O}^\dagger$ has 2^{n-1} lex-maximal consistent subontologies. Note also that the time complexity for lexicographic inference in propositional logic is Δ^p_2-complete [5], i.e. exactly in polynomial calls to an NP oracle. It is not desirable to compute all lex-maximal consistent subontologies before checking lex-consistent consequences, because such computation needs up to exponential calls to a $\mathcal{SHIQ}$ reasoner, where each call is worst-case NP-complete in data complexity [14].

To obtain a worst-case optimal method for computing lexicographic inference in $\mathcal{SHIQ}$, we consider transforming $\mathcal{SHIQ}$ to propositional logic. $\mathcal{SHIQ}$ is a subset of first-order logic. The hardness for transforming $\mathcal{SHIQ}$ to propositional logic lies on handling function symbols. Though the KAON2 transformation [14,20] can get rid of function symbols and obtain an equisatisfiable DATALOG$^\vee$ [9] program from a $\mathcal{SHIQ}$ ontology, it does not maintain the correspondence between resulting DATALOG$^\vee$ rules and original axioms in the input ontology. We therefore extend the KAON2 transformation to maintain such correspondence by introducing new nullary atoms one-to-one corresponding to axioms in the original ontology. Afterwards, we ground the transformed DATALOG$^\vee$ program and apply current powerful SAT solvers to compute lexicographic inference. In this way we obtain a worst-case optimal method in data complexity. There remains a practical problem in such method, i.e. SAT solvers lack of scalability for handling large propositional programs. To tackle this problem, we partition the transformed propositional program so that we can apply SAT solvers to handle much smaller subprograms.

Our approach is outlined as follows. Let $\mathcal{O}$ be a (possibly inconsistent) stratified $\mathcal{SHIQ}$ ontology. We first consider the basic case, i.e. checking if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ for $A(a)$ an atomic concept membership axiom, then consider the checking of other consequences. The basic case is addressed in two phases. In phase 1 (subsection 5.1), we compute a DATALOG$^\vee$ program $\mathcal{R}(\mathcal{O})$, called the *repair program* of $\mathcal{O}$, by using the extended KAON2 transformation. In phase 2 (subsection 5.2), we first ground $\mathcal{R}(\mathcal{O})$ to $\mathrm{GR}(\mathcal{O})$, then treat the problem of deciding if $\mathcal{O} \vdash_{cons}^{lex} A(a)$ as a set of satisfiability problems over $\mathrm{GR}(\mathcal{O})$ and solve them by polynomial calls to a SAT solver; in this phase we also exploit partition-based optimizations (subsection 5.3). The problem of checking other consequences is first reduced to the basic case, then solved in the same way (subsection 5.4).

5.1 Computing the Repair Program

We associate each axiom $ax \in \mathcal{O}$ with a new nullary *decision atom* $\hbar_{ax}$, such that the truth value of $\hbar_{ax}$ determines the existence of ax in $\mathcal{O}$. It should be noted that $S \sqsubseteq R$ and $Inv(S) \sqsubseteq Inv(R)$ (resp. $Trans(R)$ and $Trans(Inv(R))$) are treated as the same axiom and thus associated with the same decision atom, because they are assumed present or absent together (see this assumption in Preliminaries). Let X be the set of decision atoms wrt $\mathcal{O}$, i.e., $X = \{\hbar_{ax} \mid ax \in \mathcal{O}\}$. We extend the KAON2 transformation [14,20] to compile a DATALOG$^\vee$ program $\mathcal{R}(\mathcal{O})$ (called the *repair program* of $\mathcal{O}$), such that for any truth assignment ϕ_X on X, the *reduction* of $\mathcal{O}$ wrt ϕ_X (i.e. $\mathcal{O} \downarrow \phi_X$, see Definition 2) is satisfiable iff the *reduction* of $\mathcal{R}(\mathcal{O})$ wrt ϕ_X (i.e. $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$, see Definition 3) is satisfiable. Simply speaking, $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$ is a DATALOG$^\vee$ program without atoms in X. The relationship between $\mathcal{O} \downarrow \phi_X$ and $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$ implies a correspondence between lex-maximal consistent subontologies of $\mathcal{O}$ and certain optimal models of $\mathcal{R}(\mathcal{O})$, where the optimality is defined over X.

Definition 2. Let $\mathcal{O}$ be a $\mathcal{SHIQ}$ ontology and ϕ_X a truth assignment on the set X of decision atoms. The *reduction* of $\mathcal{O}$ wrt ϕ_X, written $\mathcal{O} \downarrow \phi_X$, is a subontology obtained from $\mathcal{O}$ by deleting each axiom ax such that $\phi_X(\hbar_{ax}) = 1$.

Definition 3. Let P be a logic program, i.e. a set of rules (or clauses), X be a set of ground atoms that only occur in rule heads (or only occur positively) in P, and ϕ_X be a truth assignment on X. The *reduction* of P wrt ϕ_X, written $P \downarrow \phi_X$, is a logic program obtained from P by deleting each rule (or clause) in P that has a head atom (or positive atom) $\alpha \in X$ such that $\phi_X(\alpha) = 1$, and by removing any ground atom $\alpha \in X$ from remaining rules (or clauses).

The compilation of $\mathcal{R}(\mathcal{O})$ roughly consists of three steps.

In step 1, we convert $\mathcal{SHIQ}$ to $\mathcal{ALCHIQ}$, i.e. $\mathcal{SHIQ}$ without transitive roles. In the original KAON2 transformation, a $\mathcal{SHIQ}$ ontology $\mathcal{O}$ is converted to an equisatisfiable $\mathcal{ALCHIQ}$ ontology $\Omega(\mathcal{O})$ by replacing each transitivity axioms $Trans(S)$ with all axioms of the form $\forall R.C \sqsubseteq \forall S.(\forall S.C)$, for each R having S as a subrole and each concept C occurring in $\mathcal{O}$. In order to make each translated clause in step 2 be associated with a single decision atom, we remove from $\Omega(\mathcal{O})$ those axioms added to $\mathcal{O}$ of the form $\forall R.C \sqsubseteq \forall S.(\forall S.C)$ where $R \neq S$, which are actually redundant axioms. By $\Omega^-(\mathcal{O})$ we denote the $\mathcal{ALCHIQ}$ ontology converted from $\mathcal{O}$ by replacing each transitivity axioms $Trans(S)$ with all axioms of the form $\forall S.C \sqsubseteq \forall S.(\forall S.C)$, for each concept C occurring in $\mathcal{O}$. The following lemma shows the correspondence between $\mathcal{O}$ and $\Omega^-(\mathcal{O})$.

Lemma 1 ([6]). *A $\mathcal{SHIQ}$ ontology $\mathcal{O}$ is satisfiable iff $\Omega^-(\mathcal{O})$ is satisfiable.* □

In step 2, we translate each axiom in $\Omega^-(\mathcal{O})$ to a set of clauses via the well-known *structural transformation* [25,14,20], mapping to first-order formulas, Skolemization and rewriting into conjunctive normal form. By $\mathrm{Cls}(ax)$ we denote the set of clauses obtained from axiom ax via the above translation process (cf. [6] for details about the translation process). For an axiom $ax \in \Omega^-(\mathcal{O})$ and a clause $cl \in \mathrm{Cls}(ax)$, by $\mathrm{em}(cl, ax)$ we denote the modified clause of cl into which a decision atom about ax is embedded: if $ax \in \Omega^-(\mathcal{O}) \setminus \mathcal{O}$, ax must be of the form $\forall S.C \sqsubseteq \forall S.(\forall S.C)$, then $\mathrm{em}(cl, ax) \overset{def}{=} cl \vee \hbar_{Trans(S)}$; otherwise, $\mathrm{em}(cl, ax) \overset{def}{=} cl \vee \hbar_{ax}$. By $\Xi(\mathcal{O})$ we denote $\mathrm{Cls}_{N_R} \cup \bigcup_{ax \in \Omega^-(\mathcal{O})} \{\mathrm{em}(cl, ax) \mid cl \in \mathrm{Cls}(ax)\}$, where the clause set $\mathrm{Cls}_{N_R} = \{\neg R(x, y) \vee R^-(y, x), \neg R^-(x, y) \vee R(y, x) \mid R \in N_R\}$ with x and y variables is introduced for mapping $\Omega^-(\mathcal{O})$ to first-order logic. The following lemma shows the correspondence between $\mathcal{O}$ and $\Xi(\mathcal{O})$.

Lemma 2 ([6]). *For any truth assignment ϕ_X on the set of decision atoms $X = \{\hbar_{ax} \mid ax \in \mathcal{O}\}$, $\mathcal{O} \downarrow \phi_X$ is satisfiable iff $\Xi(\mathcal{O}) \downarrow \phi_X$ is satisfiable.* □

Example 2. Let $\mathcal{O} = (\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4)$, where $\mathcal{O}_1 = \{ax_1 = A(a), ax_2 = B(b), ax_3 = \leq_1 T.\top(a)\}$, $\mathcal{O}_2 = \{ax_4 = T(a, b), ax_5 = T(a, c), ax_6 = b \not\approx c\}$, $\mathcal{O}_3 = \{ax_7 = A \sqsubseteq \exists R.B, ax_8 = \exists S.B \sqsubseteq \neg A\}$ and $\mathcal{O}_4 = \{ax_9 = R \sqsubseteq S\}$. Then $\Xi(\mathcal{O})$ consists of the following clauses. Note that the clauses on R^- and S^- are removed from $\Xi(\mathcal{O})$, because neither R^- nor S^- occurs in $\mathcal{O}$ and the removal does not affect the satisfiability of $\Xi(\mathcal{O}) \downarrow \phi_X$ for any truth assignment ϕ_X on $X = \{\hbar_{ax_1}, \ldots, \hbar_{ax_9}\}$.

$cl_1: A(a) \vee \hbar_{ax_1}$. $\quad cl_2: B(b) \vee \hbar_{ax_2}$. $\quad cl_3: Q_1(a) \vee \hbar_{ax_3}$.
$cl_4: y_1 \approx y_2 \vee \neg Q_1(x) \vee \neg T(x, y_1) \vee \neg T(x, y_2) \vee \hbar_{ax_3}$.
$cl_5: T(a, b) \vee \hbar_{ax_4}$. $\quad cl_6: T(a, c) \vee \hbar_{ax_5}$. $\quad cl_7: \neg(b \approx c) \vee \hbar_{ax_6}$.
$cl_8: R(x, f(x)) \vee \neg A(x) \vee \hbar_{ax_7}$. $\quad cl_9: B(f(x)) \vee \neg A(x) \vee \hbar_{ax_7}$.
$cl_{10}: \neg A(x) \vee \neg S(x, y) \vee \neg B(y) \vee \hbar_{ax_8}$. $\quad cl_{11}: S(x, y) \vee \neg R(x, y) \vee \hbar_{ax_9}$. □

In step 3, we extend the Basic Superposition ($\mathcal{BS}$) calculus [2,21] adapted in the KAON2 transformation [14,20] by considering decision atoms in the term ordering, i.e. assigning a lower precedence to decision atoms than other function symbols, constants and predicates. Then, we compute a DATALOG$^\vee$ program, i.e. the repair program $\mathcal{R}(\mathcal{O})$, from $\Xi(\mathcal{O})$ analogously as in the KAON2 transformation. The computation of $\mathcal{R}(\mathcal{O})$ is through saturating the set of translated clauses in $\Xi(\mathcal{O})$ that have variables, eliminating function symbols in the saturated set, removing irrelevant clauses and converting clauses to rules. Such computation is very technical, so we do not give details here but refer the reader to our technical report [6]. Recall that a DATALOG$^\vee$ program consists of rules R of the form $A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m$, where all A_i and B_i are atoms; the set $\mathrm{head}(R) = \{A_1, \ldots, A_n\}$ is called the *head* of R; the set $\mathrm{body}(R) = \{B_1, \ldots, B_m\}$ is called the *body* of R.

Example 3 (Example 2 continued). The repair program $\mathcal{R}(\mathcal{O})$ is constructed as follows. First, $\Xi(\mathcal{O})$ is separated into the set of clauses having variables $\Xi_{var}(\mathcal{O}) = \{cl_4, cl_8, cl_9, cl_{10}, cl_{11}\}$ and the set of variable-free clauses $\Xi_{con}(\mathcal{O}) = \{cl_1, cl_2, cl_3, cl_5, cl_6, cl_7\}$. Second, $\Xi_{var}(\mathcal{O})$ is saturated by our extended $\mathcal{BS}$ calculus, yielding the following new clauses (the notation $\mathrm{R}(cl_i, cl_j)$ means that a clause is derived by resolving clauses cl_i and cl_j).

$cl_{12}: S(x, f(x)) \vee \neg A(x) \vee \hbar_{ax_7} \vee \hbar_{ax_9}.$ $\quad$ R(cl_8, cl_{11})
$cl_{13}: \neg A(x) \vee \neg B(f(x)) \vee \hbar_{ax_7} \vee \hbar_{ax_8} \vee \hbar_{ax_9}.$ $\quad$ R(cl_{12}, cl_{10})
$cl_{14}: \neg A(x) \vee \hbar_{ax_7} \vee \hbar_{ax_8} \vee \hbar_{ax_9}.$ $\quad$ R(cl_{13}, cl_9)

Third, the clauses containing function symbols are mapped to function-free clauses given below, where each mapped clause is associated with the original sequence number. The function term $f(x)$ is mapped to a new variable x_f; meanwhile, a new negative literal $\neg S_f(x, x_f)$ is added to the mapped clause to make it *safe*, i.e. every variable occurring in positive literals occurs in some negative literal as well.

$cl_8: \neg S_f(x, x_f) \vee R(x, x_f) \vee \neg A(x) \vee \hbar_{ax_7}.$
$cl_9: \neg S_f(x, x_f) \vee B(x_f) \vee \neg A(x) \vee \hbar_{ax_7}.$
$cl_{12}: \neg S_f(x, x_f) \vee S(x, x_f) \vee \neg A(x) \vee \hbar_{ax_7} \vee \hbar_{ax_9}.$
$cl_{13}: \neg S_f(x, x_f) \vee \neg A(x) \vee \neg B(x_f) \vee \hbar_{ax_7} \vee \hbar_{ax_8} \vee \hbar_{ax_9}.$

Fourth, cl_{13} and cl_{12} are in turn detected to be irrelevant and removed. Finally, remaining clauses, together with ground clauses instantiated from $S_f(x, x_f)$, are translated into rules given below, yielding $\mathcal{R}(\mathcal{O}) = \{R_1, \dots, R_{15}\}$.

$R_1: A(a) \vee \hbar_{ax_1}. \quad R_2: B(b) \vee \hbar_{ax_2}. \quad R_3: Q_1(a) \vee \hbar_{ax_3}.$
$R_4: y_1 \approx y_2 \vee \hbar_{ax_3} \leftarrow Q_1(x), T(x, y_1), T(x, y_2).$
$R_5: T(a, b) \vee \hbar_{ax_4}. \quad R_6: T(a, c) \vee \hbar_{ax_5}. \quad R_7: \hbar_{ax_6} \leftarrow b \approx c.$
$R_8: R(x, x_f) \vee \hbar_{ax_7} \leftarrow A(x), S_f(x, x_f). \quad R_9: B(x_f) \vee \hbar_{ax_7} \leftarrow A(x), S_f(x, x_f).$
$R_{10}: \hbar_{ax_8} \leftarrow A(x), S(x, y), B(y). \quad R_{11}: S(x, y) \vee \hbar_{ax_9} \leftarrow R(x, y).$
$R_{12}: \hbar_{ax_7} \vee \hbar_{ax_8} \vee \hbar_{ax_9} \leftarrow A(x).$
$R_{13}: S_f(a, a_f). \quad R_{14}: S_f(b, b_f). \quad R_{15}: S_f(c, c_f).$ □

The following theorem shows the equisatisfiability between $\mathcal{O} \downarrow \phi_X$ and $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$ for an arbitrary truth assignment ϕ_X on X. Note that when $\mathcal{O}$ is a stratified ontology, $\mathcal{R}(\mathcal{O})$ is not stratified according to the stratification of $\mathcal{O}$. Instead, we associate the stratification of $\mathcal{O}$ with the stratification of decision atoms (see Theorem 3).

Theorem 2 ([6]). *Let $\mathcal{O}$ be a $\mathcal{SHIQ}$ ontology and X the set of decision atoms. Then: (1) For any truth assignment ϕ_X on X, $\mathcal{O} \downarrow \phi_X$ is satisfiable iff $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$ is satisfiable; (2) The number of atoms in each rule in $\mathcal{R}(\mathcal{O})$ is at most polynomial, the number of rules in $\mathcal{R}(\mathcal{O})$ is at most exponential in the size of $\mathcal{O}$, and $\mathcal{R}(\mathcal{O})$ can be computed in time exponential in the size of $\mathcal{O}$, for unary coding of numbers in input.*

Proof sketch. (For Claim 1) We can prove analogously as in Chapter 7 of [20] that, for any truth assignment ϕ_X on X, $\Xi(\mathcal{O}) \downarrow \phi_X$ and $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$ are equisatisfiable. So by Lemma 2, $\mathcal{O} \downarrow \phi_X$ and $\mathcal{R}(\mathcal{O}) \downarrow \phi_X$ are equisatisfiable. (For Claim 2) The proof is by considering the maximum number of each type of clauses that can be derived by our extended $\mathcal{BS}$ calculus, analogously as in Lemma 5.3.10 and Theorem 5.4.8 of [20]. □

5.2 Checking the Basic Consequences

To decide if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ for an atomic concept membership axiom $A(a)$, we establish, by Claim 1 of Theorem 2, a correspondence between X-lex-minimal models of the repair program $\mathcal{R}(\mathcal{O})$ and lex-maximal consistent subontologies of $\mathcal{O}$ (see Theorem 3). This correspondence implies that the problem of deciding if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ can be reduced to a certain problem of cautious reasoning over $\mathcal{R}(\mathcal{O})$ (see Theorem 4), solved by a set of satisfiability tests (see Theorem 5).

Definition 4. For a logic program P and a stratification $X = (X_1, \ldots, X_n)$ of some ground atoms in P, a model M of P is called an X*-lex-minimal model* of P if for any model M' of P, $(M \cap X_1, \ldots, M \cap X_n) \leq^{lex} (M' \cap X_1, \ldots, M' \cap X_n)$.

Theorem 3 ([6]). *Let $\mathcal{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$ be a stratified $\mathcal{SHIQ}$ ontology, and $X = (X_1, \ldots, X_n)$ be the corresponding stratification of decision atoms in $\mathcal{R}(\mathcal{O})$, i.e., $X_i = \{\hbar_{ax} \mid ax \in \mathcal{O}_i\}$ for $i = 1, \ldots, n$. Then: (1) If M is an X-lex-minimal model of $\mathcal{R}(\mathcal{O})$, then $\mathcal{O} \setminus \{ax \mid \hbar_{ax} \in M \cap X\}$ is a lex-maximal consistent subontology of $\mathcal{O}$; (2) If $\mathcal{O}' = (\mathcal{O}'_1, \ldots, \mathcal{O}'_n)$ is a lex-maximal consistent subontology of $\mathcal{O}$, then $\mathcal{R}(\mathcal{O})$ has an X-lex-minimal model M such that for each i, $M \cap X_i = \{\hbar_{ax} \mid ax \in \mathcal{O}_i \setminus \mathcal{O}'_i\}$.* □

Theorem 4 ([6]). *$\mathcal{O} \vdash^{lex}_{cons} A(a)$ iff $A(a)$ is in all X-lex-minimal models of $\mathcal{R}(\mathcal{O})$.* □

Suppose $X = (X_1, \ldots, X_n)$. It is easy to see that $(|M \cap X_1|, \ldots, |M \cap X_n|)$ is the same for every X-lex-minimal model M of $\mathcal{R}(\mathcal{O})$. By $\mathrm{lmw}(\mathcal{R}(\mathcal{O}), X) = (|M \cap X_1|, \ldots, |M \cap X_n|)$ we denote the unique *lex-minimal weight vector* of every X-lex-minimal model M of $\mathcal{R}(\mathcal{O})$, and by $\mathrm{lmw}_i(\mathcal{R}(\mathcal{O}), X)$ we denote the i^{th} element $|M \cap X_i|$ in $\mathrm{lmw}(\mathcal{R}(\mathcal{O}), X)$. Note that an interpretation is an X-lex-minimal model of $\mathcal{R}(\mathcal{O})$ iff it is a model of $\mathcal{R}(\mathcal{O}) \cup \{\sum_{\hbar_{ax} \in X_i} \mathrm{assign}(\hbar_{ax}) \leq \mathrm{lmw}_i(\mathcal{R}(\mathcal{O}), X) \mid 1 \leq i \leq n\}$, where $\mathrm{assign}(\alpha)$ denotes the 0-1 truth value of α. So Theorem 4 can be reformulated into the following theorem.

Theorem 5. *$\mathcal{O} \vdash^{lex}_{cons} A(a)$ iff $\mathcal{R}(\mathcal{O}) \cup \{\sum_{\hbar_{ax} \in X_i} \mathrm{assign}(\hbar_{ax}) \leq \mathrm{lmw}_i(\mathcal{R}(\mathcal{O}), X) \mid 1 \leq i \leq n\} \cup \{\leftarrow A(a)\}$ is unsatisfiable.* □

In order to apply Theorem 5 to check if $\mathcal{O} \vdash^{lex}_{cons} A(a)$, we need to compute lmw $(\mathcal{R}(\mathcal{O}), X)$. By Definition 4, it can be seen that $\mathrm{lmw}_i(\mathcal{R}(\mathcal{O}), X)$ is the minimum number v such that $P_i(v)$ is satisfiable, where

$$P_i(v) = \mathcal{R}(\mathcal{O}) \cup \{ \sum_{\hbar_{ax} \in X_j} \mathrm{assign}(\hbar_{ax}) \leq \mathrm{lmw}_j(\mathcal{R}(\mathcal{O}), X) \mid 1 \leq j < i\} \cup \\ \{ \sum_{\hbar_{ax} \in X_i} \mathrm{assign}(\hbar_{ax}) \leq v\} \cup \{\mathrm{assign}(\hbar_{ax}) = 0 \mid \hbar_{ax} \in \bigcup_{j=i+1}^{n} X_j\}. \tag{1}$$

I.e., $\mathrm{lmw}_i(\mathcal{R}(\mathcal{O}), X)$ is defined based on $\mathrm{lmw}_j(\mathcal{R}(\mathcal{O}), X)$ for all $j < i$, and can be computed one by one from $i = 1$ to n.

To check the satisfiability of a logic program of the form in Theorem 5 or Formula (1), we need to handle *Pseudo-Boolean constraints* (PB-constraints) of the form $\sum_i c_i \cdot \mathrm{assign}(x_i) \leq d$ with constants $c_i, d \in \mathbf{Z}$ and variables $x_i \in \{0, 1\}$, where

Z denotes the integer domain. The satisfiability problem with PB-constraints has been well studied in the SAT community. It can be either solved by standard SAT solvers after translating PB-constraints to SAT clauses [8], or solved by extended SAT solvers that support PB-constraints natively, such as PUEBLO [30].

There are some issues for applying SAT solvers: they work on propositional programs only and do not distinguish equational atoms (of the form $a \approx b$) from other atoms. To treat the equality predicate $\approx$, which is interpreted as a *congruence relation* in $\mathcal{SHIQ}$, as an ordinary predicate, we use a well-known transformation from [10]. For a DATALOG$^\vee$ program P, let $P_\approx$ denote the logic program consisting of the rules stating that $\approx$ is *reflexive*, *symmetric* and *transitive*, as well as the *replacement rules* of the form "$T(x_1, \ldots, y_i, \ldots, x_n) \leftarrow T(x_1, \ldots, x_i, \ldots, x_n), x_i \approx y_i$", instantiated for each predicate T in P other than $\approx$ and each position i. Then, appending $P_\approx$ to P allows to treat $\approx$ as an ordinary predicate.

To ground $\mathcal{R}(\mathcal{O}) \cup \mathcal{R}(\mathcal{O})_\approx$, we apply the disk-based grounding (DBG) technique in [7]. Basically, the DBG technique instantiates rules in a traditional bottom-up iterative manner, applying a SQL engine to maintain ground atoms in instantiated rules. By $\mathrm{GR}(\mathcal{O})$ we denote the propositional program grounded from $\mathcal{R}(\mathcal{O}) \cup \mathcal{R}(\mathcal{O})_\approx$ by applying the DBG technique. There exists a bijection between the set of minimal models of $\mathrm{GR}(\mathcal{O})$ and that of $\mathcal{R}(\mathcal{O}) \cup \mathcal{R}(\mathcal{O})_\approx$, where two corresponding minimal models coincide except on some ground atoms not over concept/role names. The following corollary is an immediate consequence that follows from Theorem 4 and Theorem 5, where $\mathrm{lmw}_i(\mathrm{GR}(\mathcal{O}), X) = \mathrm{lmw}_i(\mathcal{R}(\mathcal{O}), X)$ for each i.

Corollary 1 ([6]). *$\mathcal{O} \vdash^{lex}_{cons} A(a)$ iff $A(a)$ is in all X-lex-minimal models of $\mathrm{GR}(\mathcal{O})$, i.e., $\mathrm{GR}(\mathcal{O}) \cup \{\sum_{\hbar_{ax} \in X_i} \mathrm{assign}(\hbar_{ax}) \leq \mathrm{lmw}_i(\mathrm{GR}(\mathcal{O}), X) \mid 1 \leq i \leq n\} \cup \{\leftarrow A(a)\}$ is unsatisfiable.* □

Example 4 (Example 3 continued). By applying the DBG technique, $\mathcal{R}(\mathcal{O}) \cup \mathcal{R}(\mathcal{O})_\approx$ is grounded to $\mathrm{GR}(\mathcal{O}) = \{r_1, \ldots, r_{16}\}$ given below. Note that some optimizations on instantiating $\mathcal{R}(\mathcal{O})_\approx$ are used in this example, cf. [6].

$$
\begin{array}{l}
r_1 : A(a) \vee \hbar_{ax_1}. \quad r_2 : B(b) \vee \hbar_{ax_2}. \quad r_3 : Q_1(a) \vee \hbar_{ax_3}.\\
r_4 : T(a,b) \vee \hbar_{ax_4}. \quad r_5 : T(a,c) \vee \hbar_{ax_5}. \quad r_6 : \hbar_{ax_6} \leftarrow b \approx c.\\
r_7 : b \approx c \vee \hbar_{ax_3} \leftarrow Q_1(a), T(a,b), T(a,c). \quad r_8 : R(a,a_f) \vee \hbar_{ax_7} \leftarrow A(a).\\
r_9 : B(a_f) \vee \hbar_{ax_7} \leftarrow A(a). \quad r_{10} : S(a,a_f) \vee \hbar_{ax_9} \leftarrow R(a,a_f).\\
r_{11} : \hbar_{ax_8} \leftarrow A(a), S(a,a_f), B(a_f). \quad r_{12} : \hbar_{ax_7} \vee \hbar_{ax_8} \vee \hbar_{ax_9} \leftarrow A(a).\\
r_{13} : T(a,c) \leftarrow T(a,b), b \approx c. \quad r_{14} : T(a,b) \leftarrow T(a,c), b \approx c.\\
r_{15} : B(c) \leftarrow B(b), b \approx c. \quad r_{16} : B(b) \leftarrow B(c), b \approx c.
\end{array}
$$

It can be computed by Formula (1), where $\mathcal{R}(\mathcal{O})$ is replaced with $\mathrm{GR}(\mathcal{O})$, that lmw $(\mathrm{GR}(\mathcal{O}), X) = (0, 1, 0, 1)$. Consider deciding if $\mathcal{O} \vdash^{lex}_{cons} A(a)$. The satisfiability of $\Pi = \mathrm{GR}(\mathcal{O}) \cup \{\mathrm{assign}(\hbar_{ax_1}) + \mathrm{assign}(\hbar_{ax_2}) + \mathrm{assign}(\hbar_{ax_3}) \leq 0, \mathrm{assign}(\hbar_{ax_4}) + \mathrm{assign}(\hbar_{ax_5}) + \mathrm{assign}(\hbar_{ax_6}) \leq 1, \mathrm{assign}(\hbar_{ax_7}) + \mathrm{assign}(\hbar_{ax_8}) \leq 0, \mathrm{assign}(\hbar_{ax_9}) \leq 1\} \cup \{\leftarrow A(a)\}$ is tested. It is easy to see that Π is unsatisfiable, so $\mathcal{O} \vdash^{lex}_{cons} A(a)$. □

Consider the time complexity for checking if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ in terms of *data complexity*, i.e. the complexity measured as a function of $|\mathcal{O}_\mathcal{A}|$. Since saturating the clause set $\Xi(\mathcal{O})$ is only performed over the subset transformed from terminological axioms, by viewing the considering ontology in Claim 2 of Theorem [2] as the terminology of $\mathcal{O}$ only, the saturation of $\Xi(\mathcal{O})$ is accomplished in constant time and the number of rules in $\mathcal{R}(\mathcal{O})$ is polynomial in $|\mathcal{O}_\mathcal{A}|$. For every rule $R \in \mathcal{R}(\mathcal{O})$, the number of variables in R is bounded by a constant, so there are at most polynomial ground instances of R in $\mathrm{GR}(\mathcal{O})$. It follows that the number of rules in $\mathrm{GR}(\mathcal{O})$ is polynomial in $|\mathcal{O}_\mathcal{A}|$. In addition, the computation of $\mathrm{lmw}(\mathrm{GR}(\mathcal{O}), X)$ needs at most $|X|$ satisfiability tests, so checking if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ is accomplished in $|X| + 1$ satisfiability tests over $\mathrm{GR}(\mathcal{O})$. Since the satisfiability problem with PB-constraints is NP-complete, the problem of deciding if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ is thus in Δ^p_2 in data complexity. Note that the addressing problem is also Δ^p_2-hard (see the following theorem), so our proposed method is worst-case optimal in data complexity.

Theorem 6 ([6]). *For a stratified $\mathcal{SHIQ}$ ontology $\mathcal{O}$ and an atomic concept membership axiom $A(a)$, the problem of deciding if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ is data complete for Δ^p_2.*

Proof sketch. The Δ^p_2 membership has been shown by our proposed method. The Δ^p_2 hardness is proved by a polynomial time reduction from the following Δ^p_2-complete problem [16]: "given a satisfiable clause set $C = \{C_1, \ldots, C_m\}$ on $X = \{x_1, \ldots, x_n\}$, decide whether the lexicographically minimum truth assignment ϕ^m_X satisfying C fulfills $\phi^m_X(x_n) = 1$". □

5.3 Partition-Based Optimizations

To make the above method more scalable, we adapt the partitioning technique in [7] to decompose $\mathrm{GR}(\mathcal{O})$ into disjoint subprograms before checking if $\mathcal{O} \vdash^{lex}_{cons} A(a)$. By that means, the satisfiability test in Corollary [1] can be performed over a small relevant part of $\mathrm{GR}(\mathcal{O})$.

Let $\mathrm{atoms}(P)$ denote the set of ground atoms in a propositional program P. By applying the partitioning algorithm given in [7], which can also be found in our technical report [6], we decompose $\mathrm{GR}(\mathcal{O})$ into a set of disjoint subprograms $\{\mathrm{GR}_i(\mathcal{O})\}_{1 \le i \le p}$. The decomposition is through sequentially accessing $\mathrm{GR}(\mathcal{O})$ at most $\min(|\mathrm{GR}(\mathcal{O})|, |\mathrm{atoms}(\mathrm{GR}(\mathcal{O}))|)$ times, where $|\mathrm{GR}(\mathcal{O})|$ denotes the number of rules in $\mathrm{GR}(\mathcal{O})$ and $|\mathrm{atoms}(\mathrm{GR}(\mathcal{O}))|$ denotes the number of atoms in $\mathrm{GR}(\mathcal{O})$. The resulting set has the following properties: (1) $\mathrm{GR}_j(\mathcal{O})$ and $\mathrm{GR}_k(\mathcal{O})$ have no common ground atoms for any $j \neq k$; (2) each $\mathrm{GR}_i(\mathcal{O})$ is a connected component, i.e., for any two ground atoms α_1 and α_2 occurring in $\mathrm{GR}_i(\mathcal{O})$, there exists a sequence of ground atoms $\beta_1 = \alpha_1, \ldots, \beta_n = \alpha_2$ such that β_k and β_{k+1} occur together in some rule in $\mathrm{GR}_i(\mathcal{O})$ for $k = 1, \ldots, n-1$; (3) each $\mathrm{GR}_i(\mathcal{O})$ is a certain rule closure, i.e., for each rule $r \in \mathrm{GR}(\mathcal{O})$ such that $\emptyset \subset \mathrm{head}(r) \setminus X \subseteq \mathrm{atoms}(\mathrm{GR}_i(\mathcal{O}) \setminus \{r\})$, $r \in \mathrm{GR}_i(\mathcal{O})$; and (4) for each rule $r \in \mathrm{GR}(\mathcal{O}) \setminus \bigcup_{i=1}^{p} \mathrm{GR}_i(\mathcal{O})$, there exists some ground atom $\alpha \in \mathrm{head}(r)$ such that $\alpha \notin \mathrm{atoms}(\bigcup_{i=1}^{p} \mathrm{GR}_i(\mathcal{O})) \cup X$. Let $X^\dagger_i$ denote the subset of decision atoms that occur in $\mathrm{GR}_i(\mathcal{O})$. Then $X^\dagger_i$ can be written as $(X^\dagger_{i,1}, \ldots, X^\dagger_{i,n})$ based on the stratification of $\mathcal{O}$, i.e., $X^\dagger_{i,j} = X^\dagger_i \cap \{\hbar_{ax} \mid ax \in \mathcal{O}_j\}$ for $j = 1, \ldots, n$.

By using the partitioning results, we develop a novel algorithm for checking if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ (cf. [6] for more explanations on this algorithm). Let $\mathrm{GR}_0(\mathcal{O}) = \mathrm{GR}(\mathcal{O}) \setminus \bigcup_{i=1}^{p} \mathrm{GR}_i(\mathcal{O})$ and $M_0 = \bigcup_{r \in \mathrm{GR}_0(\mathcal{O})} \mathrm{head}(r) \setminus (\mathrm{atoms}(\bigcup_{i=1}^{p} \mathrm{GR}_i(\mathcal{O})) \cup X)$.

Then by Property (4) in the previous paragraph, $M_0 \cap \text{head}(r) \neq \emptyset$ for all rules $r \in \text{GR}_0(\mathcal{O})$. Let X' be the set of decision atoms not occurring in $\bigcup_{i=1}^{p} \text{GR}_i(\mathcal{O})$, i.e., $X' = X \setminus \text{atoms}(\bigcup_{i=1}^{p} \text{GR}_i(\mathcal{O}))$. Let $\text{GR}'_0(\mathcal{O})$ be obtained from $\text{GR}_0(\mathcal{O})$ by deleting all decision atoms in X', i.e., $\text{GR}'_0(\mathcal{O}) = \{\bigvee \text{head}(r) \setminus X' \leftarrow \bigwedge \text{body}(r) \mid r \in \text{GR}_0(\mathcal{O})\}$. Let $\text{MM}_0(\mathcal{O})$ denote the unique minimal model of $\{r \in \text{GR}'_0(\mathcal{O}) \mid |\text{head}(r)| = 1\}$. For every X-lex-minimal model M of $\text{GR}(\mathcal{O})$, M does not contain any decision atom in X', otherwise $M' = (M \cap \text{atoms}(\bigcup_{i=1}^{p} \text{GR}_i(\mathcal{O}))) \cup M_0$ will be a model of $\text{GR}(\mathcal{O})$ such that $(M' \cap X_1, \ldots, M' \cap X_n) <^{lex} (M \cap X_1, \ldots, M \cap X_n)$, contracting that M is an X-lex-minimal model of $\text{GR}(\mathcal{O})$. Hence, $M \cap \text{atoms}(\text{GR}'_0(\mathcal{O}))$ is a model of $\text{GR}'_0(\mathcal{O})$. It follows that $\text{MM}_0(\mathcal{O}) \subseteq M$. Let $\text{GR}^r_0(\mathcal{O})$ be obtained from $\text{GR}'_0(\mathcal{O})$ by deleting each rule that has at least one head atom in $\text{MM}_0(\mathcal{O})$, and by removing all atoms in $\text{MM}_0(\mathcal{O})$ from the remaining rules. Then, the satisfiability test in Corollary 1 can be performed over a subprogram of $\text{GR}(\mathcal{O})$, extracted from $\text{GR}_i(\mathcal{O})_{1 \leq i \leq p}$, $\text{GR}^r_0(\mathcal{O})$ or $\text{MM}_0(\mathcal{O})$, as shown below, where $\text{lmw}_i(\text{GR}_k(\mathcal{O}), X^\dagger_k)$ can be computed over $\text{GR}_k(\mathcal{O})$ in the same way as computing $\text{lmw}_i(\mathcal{R}(\mathcal{O}), X)$, cf. Formula (1).

1. In case $A(a) \in \text{MM}_0(\mathcal{O})$, $\mathcal{O} \vdash^{lex}_{cons} A(a)$.
2. In case $A(a) \in \text{atoms}(\text{GR}_k(\mathcal{O}))$ for some $k > 0$, let $\Pi = \text{GR}_k(\mathcal{O}) \cup \{\sum_{\hbar_{ax} \in X^\dagger_{k,i}} \text{assign}(\hbar_{ax}) \leq \text{lmw}_i(\text{GR}_k(\mathcal{O}), X^\dagger_k) \mid 1 \leq i \leq n\}$. I.e., Π is $\text{GR}_k(\mathcal{O})$ together with PB-constraints encoded from the criterion for $X^\dagger_k$-lex-minimal models of $\text{GR}_k(\mathcal{O})$. Then $\mathcal{O} \vdash^{lex}_{cons} A(a)$ iff $\Pi \cup \{\leftarrow A(a)\}$ is unsatisfiable.
3. In case $A(a) \in \text{head}(r) \setminus \text{atoms}(\bigcup_{i=1}^{p} \text{GR}_i(\mathcal{O}))$ for some rule $r \in \text{GR}^r_0(\mathcal{O})$, let Π be the fixpoint of Π_t such that $\Pi_0 = \{r \in \text{GR}^r_0(\mathcal{O}) \mid \text{head}(r) \subseteq X \cup \{A(a)\}\}$, and $\Pi_t = \Pi_{t-1} \cup \{r \in \text{GR}^r_0(\mathcal{O}) \mid \text{head}(r) \subseteq \text{atoms}(\Pi_{t-1} \cup \bigcup_{i=1}^{p} \text{GR}_i(\mathcal{O}))\}$ for $t > 0$. Let $S_N = \{1 \leq i \leq p \mid \text{atoms}(\Pi) \cap \text{atoms}(\text{GR}_i(\mathcal{O})) \neq \emptyset\}$ and $\Pi' = \Pi \cup \bigcup_{k \in S_N} (\text{GR}_k(\mathcal{O}) \cup \{\sum_{\hbar_{ax} \in X^\dagger_{k,i}} \text{assign}(\hbar_{ax}) \leq \text{lmw}_i(\text{GR}_k(\mathcal{O}), X^\dagger_k) \mid 1 \leq i \leq n\})$. I.e., Π is a subprogram extracted from $\text{GR}^r_0(\mathcal{O})$; Π' is the union of Π and every $\text{GR}_k(\mathcal{O})$ that has some atoms occurring in Π, together with PB-constraints encoded from the criterion for $X^\dagger_k$-lex-minimal models of $\text{GR}_k(\mathcal{O})$. Then $\mathcal{O} \vdash^{lex}_{cons} A(a)$ iff $\Pi' \cup \{\leftarrow A(a)\}$ is unsatisfiable.
4. In other cases, $\mathcal{O} \not\vdash^{lex}_{cons} A(a)$.

Theorem 7 ([6]). *The above algorithm for checking if $\mathcal{O} \vdash^{lex}_{cons} A(a)$ is correct.* □

Example 5 (Example 4 continued). By applying the partitioning algorithm in [7], $\text{GR}(\mathcal{O})$ is decomposed into $\text{GR}_1(\mathcal{O}) = \{r_1, r_8, r_9, r_{10}, r_{11}, r_{12}\}$ and $\text{GR}_2(\mathcal{O}) = \{r_3, r_4, r_5, r_6, r_7, r_{13}, r_{14}\}$. Afterwards, the decomposition is continued to yield in turn $\text{GR}_0(\mathcal{O}) = \{r_2, r_{15}, r_{16}\}$, $\text{MM}_0(\mathcal{O}) = \{B(b)\}$ and $\text{GR}^r_0(\mathcal{O}) = \{B(c) \leftarrow b \approx c\}$. The decomposition also divides X into $X^\dagger_1 = (\{\hbar_{ax_1}\}, \emptyset, \{\hbar_{ax_7}, \hbar_{ax_8}\}, \{\hbar_{ax_9}\})$ and $X^\dagger_2 = (\{\hbar_{ax_3}\}, \{\hbar_{ax_4}, \hbar_{ax_5}, \hbar_{ax_6}\}, \emptyset, \emptyset)$. It can be computed by Formula (1), where $\mathcal{R}(\mathcal{O})$ and X are replaced with $\text{GR}_i(\mathcal{O})$ and $X^\dagger_i$ respectively, that $\text{lmw}(\text{GR}_1(\mathcal{O}), X^\dagger_1) = (0, 0, 0, 1)$ and $\text{lmw}(\text{GR}_2(\mathcal{O}), X^\dagger_2) = (0, 1, 0, 0)$.

Consider deciding if $\mathcal{O} \vdash^{lex}_{cons} A(a)$. Since $A(a) \in \text{atoms}(\text{GR}_1(\mathcal{O}))$, the satisfiability of Π is tested, where $\Pi = \text{GR}_1(\mathcal{O}) \cup \{\text{assign}(\hbar_{ax_1}) \leq 0,\ \text{assign}(\hbar_{ax_7}) +$

$\text{assign}(\hbar_{ax_8}) \leq 0, \text{assign}(\hbar_{ax_9}) \leq 1\}$. Since $\Pi \cup \{\leftarrow A(a)\}$ is unsatisfiable, we have $\mathcal{O} \vdash^{lex}_{cons} A(a)$.

Consider deciding if $\mathcal{O} \vdash^{lex}_{cons} B(c)$. Since $B(c) \in \text{head}(r) \setminus \text{atoms}(\text{GR}_1(\mathcal{O}) \cup \text{GR}_2(\mathcal{O}))$ for the unique rule $r \in \text{GR}^r_0(\mathcal{O})$, the satisfiability of $\Pi' \cup \{\leftarrow B(c)\}$ is tested, where $\Pi' = \text{GR}^r_0(\mathcal{O}) \cup \text{GR}_1(\mathcal{O}) \cup \{\text{assign}(\hbar_{ax_3}) \leq 0, \text{assign}(\hbar_{ax_4}) + \text{assign}(\hbar_{ax_5}) + \text{assign}(\hbar_{ax_6}) \leq 1\}$. Since $\Pi' \cup \{\leftarrow B(c)\}$ is satisfiable (e.g., the set $\{Q_1(a), T(a,b), \hbar_{ax_5}\}$ is a model), we have $\mathcal{O} \not\vdash^{lex}_{cons} B(c)$.

Consider deciding if $\mathcal{O} \vdash^{lex}_{cons} B(b)$. Since $B(b) \in \text{MM}_0(\mathcal{O})$, we have $\mathcal{O} \vdash^{lex}_{cons} B(b)$. Consider deciding if $\mathcal{O} \vdash^{lex}_{cons} B(a)$. Since $B(a) \notin \text{MM}_0(\mathcal{O}) \cup \text{atoms}(\text{GR}^r_0(\mathcal{O}) \cup \text{GR}_1(\mathcal{O}) \cup \text{GR}_2(\mathcal{O}))$, we have $\mathcal{O} \not\vdash^{lex}_{cons} B(a)$. □

5.4 Checking Other Consequences

To decide if $\mathcal{O} \vdash^{lex}_{cons} C(a)$ for $\mathcal{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$ and C a general concept, we consider a new ontology $\mathcal{O}' = (\{C \sqsubseteq Q\}, \mathcal{O}_1, \ldots, \mathcal{O}_n)$, where Q is a new concept name. Since the axiom $C \sqsubseteq Q$ must be in every lex-maximal consistent subontology of $\mathcal{O}'$, we have $\mathcal{O} \vdash^{lex}_{cons} C(a)$ iff $\mathcal{O}' \vdash^{lex}_{cons} Q(a)$, where the latter can be checked using the proposed method, as $Q(a)$ is an atomic concept membership axiom.

To decide if $\mathcal{O} \vdash^{lex}_{cons} C \sqsubseteq D$ for C and D general concepts, we consider the axiom $(\neg C \sqcup D)(a)$ for a a new globally unique individual. Since $C \sqsubseteq D$ is equivalent to $\top \sqsubseteq \neg C \sqcup D$ and $\mathcal{O} \vdash^{lex}_{cons} (\top \sqsubseteq \neg C \sqcup D)$ iff $\mathcal{O} \vdash^{lex}_{cons} (\neg C \sqcup D)(a)$, we have $\mathcal{O} \vdash^{lex}_{cons} C \sqsubseteq D$ iff $\mathcal{O} \vdash^{lex}_{cons} (\neg C \sqcup D)(a)$, where the latter can be checked using the method just given above.

To decide if $\mathcal{O} \vdash^{lex}_{cohe} C(a)$ or decide if $\mathcal{O} \vdash^{lex}_{cohe} C \sqsubseteq D$ for C and D general concepts, we first reduce the problem to that of checking the corresponding lex-consistent consequence by Theorem 1, then solve it in the same way.

6 Conclusion and Future Work

In this paper, we applied the lexicographic inference to reason over inconsistent DL-based ontologies and addressed the problem of checking lex-consistent (or lex-coherent) consequences of a $\mathcal{SHIQ}$ ontology. Basically, our proposed method compiles the input $\mathcal{SHIQ}$ ontology to a propositional program, so that the addressing problem is solved in polynomial calls to current powerful SAT solvers. The method is the first worst-case optimal one (in data complexity) for checking lex-consistent consequences of a $\mathcal{SHIQ}$ ontology. It performs the checking without computing any lex-maximal consistent subontology. It can also be applied to check lex-coherence consequences by first reducing the problem to that of checking lex-consistent consequences. In order to make the method more scalable, we also gave partition-based techniques to optimize the calling of SAT solvers. For future work, we plan to conduct a thorough evaluation for the proposed method and find feasible approaches to handle OWL DL ontologies.

Acknowledgments. This work is supported in part by NSFC grants 60673103 and 60721061.

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. Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W.: Basic paramodulation. Information and Computation 121(2), 172–192 (1995)
3. Benferhat, S., Cayrol, C., Dubois, D., Lang, J., Prade, H.: Inconsistency management and prioritized syntax-based entailment. In: Proc. of IJCAI 1993, pp. 640–647 (1993)
4. Benferhat, S., Dubois, D., Prade, H.: How to infer from inconsisent beliefs without revising? In: Proc. of IJCAI 1995, pp. 1449–1457 (1995)
5. Cayrol, C., Lagasquie-Schiex, M.: On the complexity of non-monotonic entailment in syntax-based approaches. In: Proc. of ECAI 1994 Workshop on Algorithms, Complexity and Commonsense Reasoning (1994)
6. Du, J., Qi, G., Shen, Y.: Lexicographical inference over inconsistent DL-based ontologies. Technical report, Institute of Software, Chinese Academy of Sciences, Beijing, China (May 2008), http://lcs.ios.ac.cn/~jfdu/papers/lexinf-tr.pdf
7. Du, J., Shen, Y.: Computing minimum cost diagnoses to repair populated DL-based ontologies. In: Proc. of WWW 2008, pp. 565–574 (2008)
8. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT 2, 1–26 (2006)
9. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Transactions on Database Systems 22(3), 364–418 (1997)
10. Fitting, M.: First-order Logic and Automated Theorem Proving, 2nd edn. Springer, New York (1996)
11. Haase, 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)
12. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL 8(3) (2000)
13. Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with inconsistent ontologies. In: Proc. of IJCAI 2005, pp. 454–459 (2005)
14. Hustadt, U., Motik, B., Sattler, U.: Reasoning in description logics by a reduction to disjunctive datalog. Journal of Automated Reasoning 39(3), 351–384 (2007)
15. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C.: Repairing unsatisfiable concepts in OWL ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 170–184. Springer, Heidelberg (2006)
16. Krentel, M.W.: The complexity of optimization problems. Journal of Computer and System Sciences 36(3), 490–509 (1988)
17. Lembo, D., Ruzzi, M.: Consistent query answering over description logic ontologies. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 194–208. Springer, Heidelberg (2007)
18. 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)
19. Meyer, T., Lee, K., Booth, R.: Knowledge integration for description logics. In: Proc. of AAAI 2005, pp. 645–650 (2005)
20. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Univesität karlsruhe, Germany (January 2006)
21. Nieuwenhuis, R., Rubio, A.: Theorem proving with ordering and equality constrained clauses. Journal of Symbolic Computation 19(4), 321–351 (1995)

22. Odintsov, S.P., Wansing, H.: Inconsistency-tolerant Description Logic: Motivation and Basic Systems. In: Trends in Logic: 50 Years of Studia Logica, pp. 301–335. Kluwer Academic Publishers, Dordrecht (2003)
23. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Proc. of WWW 2005, pp. 633–640 (2005)
24. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation (2004), http://www.w3.org/TR/owl-semantics/
25. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation 2(3), 293–304 (1986)
26. Qi, G., Liu, W., Bell, D.: A revision-based approach to handling inconsistency in description logics. Artificial Intelligence Review 26(1-2), 115–128 (2006)
27. Qi, G., Pan, J.Z., Ji, Q.: Extending description logics with uncertainty reasoning in possibilistic logic. In: Mellouli, K. (ed.) ECSQARU 2007. LNCS (LNAI), vol. 4724, pp. 828–839. Springer, Heidelberg (2007)
28. Schlobach, S.: Diagnosing terminologies. In: Proc. of AAAI 2005, pp. 670–675 (2005)
29. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Proc. of IJCAI 2003, pp. 355–362 (2003)
30. Sheini, H.M., Sakallah, K.A.: Pueblo: A hybrid pseudo-boolean SAT solver. JSAT 2, 157–181 (2006)

A Survey of Revision Approaches in Description Logics*

Guilin Qi[1] and Fangkai Yang[2]

[1] Institute AIFB
Universität Karlsruhe (TH), Germany
gqi@aifb.uni-karlsruhe.de
[2] Fangkai Yang
Multi-Agent System Lab
University of Science and Technology of China, Hefei, China
fkyang@mail.ustc.edu.cn

Abstract. Revision of a Description Logic-based ontology to incorporate newly received information consistently is an important problem for the lifecycle of ontologies. Many approaches in the theory of belief revision have been applied to deal with this problem and most of them focus on the postulates or the logical properties of a revision operator in Description Logics (DLs). However, there is no coherent view on how to characterize a revision operator in DLs. In this paper, we lay bare the assumptions underlying different approaches for revision in DLs and propose some criteria to compare them. Based on the analysis, we give our definition of a revision operator in DLs and point out some open problems.

1 Introduction

Ontologies play a crucial role for the success of the Semantic Web [3]. One of the challenging problems for the development of ontology is ontology evolution, which is defined as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes [10,27]. One of the central problems during ontology evolution is inconsistency handling. There are various forms of inconsistencies, such as structural inconsistency, logical inconsistency and user-defined inconsistency. Among them, logical inconsistency has received lots of attention, where ontologies are represented by logical theories, such as Description Logics (DLs) [6,10,11].

Theory of belief change in propositional logic or first-order logic deals with the logical inconsistency resulting from revising a knowledge base by newly received information. There are three types of belief change, i.e. *expansion*, *contraction* and *revision*. Expansion is simply to add a formula to a knowledge base; contraction requires to consistently remove a formula from a knowledge base and revision is the problem of accommodating a new formula to a knowledge base consistently. Alchourrón, Gardenfors and Markinson (AGM) propose a set of postulates to characterize each belief change operator (see [8] for their work). The application of AGM's theory to description logics is not trivial because it is based on the assumptions that generally fail for DLs [5].

* The first author is partially supported by the EU under the IST project NeOn (IST-2006-027595, http://www.neon-project.org/).

In [5,6], the basic AGM postulates for contraction are generalized to DLs and the feasibility of applying the generalized AGM theory of contraction to DLs and OWL is studied. However, they do not consider the application of AGM postulates for revision in DLs. According to [6], DL $\mathcal{SHIF}(\mathcal{D})$ and DL $\mathcal{SHOIN}(\mathcal{D})$ are not *AGM compliant*, i.e., we cannot define a contraction operator that satisfies the generalized AGM postulates in these DLs.

In AGM's theory, epistemic states are represented by belief sets, i.e., sets of formulas that are closed under logical consequence. However, in a belief set, there is no difference between the explicitly represented knowledge (i.e., those formulas that are stored in the databases) and the implicitly represented one (i.e., those formulas that are implied by the explicitly represented one). Therefore, many researchers turn to use a belief base, i.e., a finite set of formulas that may not be closed under logical consequence, to represent epistemic sates of an agent (see [7,18,14,16]). In [4], the authors argue that in the context of the ontology evolution, it is more natural to differentiate the explicit knowledge and implied one in an ontology. They then propose a set of postulates for contraction operators by dropping the assumption that the result of contraction is an ontology closed under logical consequence. They also define a set of postulates for revision. In [12], Hansson's semi-revision operator is applied to deal with the revision problem in DLs and an algorithm is given to implement the revision operator. However, semi-revision operator does not guarantee the success postulate, which says that the newly received information will be kept after revision. Therefore, in [25], two revised semi-revision operators are given to guarantee the property of (weak) success, which says that we should not remove the new information unless it leads to contradiction. The disadvantage of their approaches is that they are dependent on the syntactical forms of axioms and therefore are not fine-grained.

Most of the important work on revising DL knowledge bases is based on the AGM theory. However, compared with mature research in the area of belief revision, e.g., AGM's theory, some of them are not well justified and also deviate from the AGM paradigms, leading to a misconception that AGM theory is unsuitable for DLs. Therefore, it is important to analyze these approaches to lay bare the assumptions underlying them. The forthcoming Semantic Web constitutes an ideal application scenario for formal logic and traditional commonsense reasoning approaches. Therefore, many people from traditional knowledge representation and reasoning community are interested in applying the work on belief revision to DLs. But they may not be familiar with the literature and find it hard to know where to start. So, another motivation of this work is to report the latest progress of the work on revision in DLs and point out some future directions to facilitate their study.

In this paper, we investigate the problem of revision in description logics. We mainly consider the following questions:

– What is a revision operator in description logics? This is a fundamental question to be answered.
– What postulates should a revision operator satisfy? This is the main question that will be discussed in this paper. This question is closely related to the first one.

This paper is organized as follows. Section 2 briefly introduces Description Logics and Section 3 gives a short survey of belief revision theory. Section 4 gives a survey of existing revision approaches in description logics. We also propose some criteria to facilitate the comparison among these approaches. Finally, we conclude the paper, and give some open questions in Section 6.

2 Description Logics

We introduce some basic definitions of Description Logics (DLs). More comprehensive treatment of DLs can be found in Chapters 2 and 4 of [2].

We start from a set N_C of concept names, a set N_R of role names and a set N_I of individual names. Concepts in DLs are defined inductively with the help of a set of constructors, such as conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$) and nominal ($\{a\}$) where a is an individual name. The expressive power of a DL is determined by the constructors that are used in it.

A *terminology axiom* is an expression of the form $C \sqsubseteq D$ where C and D are concept expressions, and a *role axiom* is an expression of the form $R \sqsubseteq S$, where R and S are role expressions. A TBox, denoted by $\mathcal{T}$, is a set of terminology axioms and role axioms which are viewed as intensional description of the domain of interest. An assertional axiom is an expression of the form $C(a)$ or $R(a, b)$, where C is a concept expression, R is a role expression and a and b are individual names. An ABox, denoted by $\mathcal{A}$, is a set of assertional axioms, which are viewed as extensional information. Finally, A DL-based knowledge base (or ontology) is a pair $K = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ and $\mathcal{A}$ are a TBox and an ABox respectively.

The semantics of a DL is defined by an interpretation $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ which consists of a non-empty domain set $\triangle^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which maps from individuals, concepts and roles to elements of the domain, subsets of the domain and binary relations on the domain, respectively. Given an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ satisfies a terminology axiom $C \sqsubseteq D$ (resp., a role axiom $R \sqsubseteq S$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp., $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$). Furthermore, $\mathcal{I}$ satisfies a concept assertion $C(a)$ (resp., a role assertion $R(a, b)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$). An interpretation $\mathcal{I}$ is called a *model* of an ontology K, iff it satisfies each axiom in K. A concept name C in an ontology K, is unsatisfiable if for each model $\mathcal{I}$ of K, $C^{\mathcal{I}} = \emptyset$. An ontology K is incoherent if there exists an unsatisfiable concept name in K. An ontology K is inconsistent iff it has no model.

3 Belief Revision

We give a short survey of belief revision theory and refer the interesting reader to Chapter 8 of [28]. The purpose of this section is to provide some background knowledge for understanding the analysis of existing revision approaches in DLs. For this reason, we will focus on two lines of work: AGM belief change theory [8] and Hansson's belief dynamics theory [17].

Theory of belief revision deals with logical inconsistency resulting from revising a knowledge base by a piece of newly received information. In their pioneer work in [1], Alchourrón, Gardenfors and Markinson (AGM) propose a set of postulates to characterize a rational revision operator. AGM' framework is built on a formal language L

in a logic which is identified by its consequence operation Cn, where L is assumed to be closed under all the boolean connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication), and Cn should satisfy the following conditions: for any set of formulas X and Y, $X \subseteq Cn(X)$, $Cn(X) = Cn(Cn(X))$ and $Cn(X) \subseteq Cn(Y)$ whenever $X \subseteq Y$. For example, L can be based on first order logic or classical propositional logic. The symbols $\top$ and $\bot$ will be used to denote respectively an arbitrary tautology and contradiction of L. In AGM's work, beliefs of an agent are represented by a set of formulas closed under logical consequence, called a belief set, i.e., a set K such that $K = Cn(K)$. A revision operator is an operation $*$ that maps a belief set K and a formula ϕ to a belief set $K * \phi$. Additional constraints should be imposed on revision operator $*$ to capture the notion of *rational* belief revision. One of the important constraints is the *principle of minimal change*, that is, a rational agent should change her beliefs *as little as possible* to accommodate the new information consistently. To define a rational belief revision operator, AGM propose a set of eight postulates, known as the AGM postulates for belief revision, as follows: let $K + \phi = Cn(K \cup \{\phi\})$,

$(K * 1)$ $K * \phi$ is a theory of L.
$(K * 2)$ $\phi \in K * \phi$.
$(K * 3)$ $K * \phi \subseteq K + \phi$.
$(K * 4)$ If $\neg\phi \notin K$ then $K + \phi \subseteq K * \phi$.
$(K * 5)$ If ϕ is consistent then $K * \phi$ is also consistent.
$(K * 6)$ If $Cn(\phi) = Cn(\psi)$ then $K * \phi = K * \psi$.
$(K * 7)$ $K * (\phi \wedge \psi) \subseteq (K * \phi) + \psi$.
$(K * 8)$ If $\neg\psi \notin K * \phi$ then $(K * \phi) + \psi \subseteq K * (\phi \wedge \psi)$.

Explanation of these postulates can be found in Chapter 8 of [28].

AGM also study another type of belief change called *belief contraction*, the problem of giving up a certain belief ϕ in a belief set K. Formally, a belief contraction operator $-$ is a function mapping a theory K and a formula ϕ to a new theory $K - \phi$. To capture a *rational* belief contraction operator, AGM propose a set of postulates, known as AGM postulates for belief contraction:

$(K - 1)$ $K - \phi$ is a theory of L.
$(K - 2)$ $K - \phi \subseteq K$.
$(K - 3)$ If $\phi \notin K$ then $K - \phi = K$.
$(K - 4)$ If $Cn(\phi) \neq Cn(\top)$ then $\phi \notin K - \phi$.
$(K - 5)$ If $\phi \in K$ then $K \subseteq (K - \phi) + \phi$.
$(K - 6)$ If $Cn(\phi) = Cn(\psi)$ then $K - \phi = K - \psi$.
$(K - 7)$ $(K - \phi) \cap (K - \psi) \subseteq K - (\phi \wedge \psi)$.
$(K - 8)$ If $\phi \notin K - (\phi \wedge \psi)$ then $K - (\phi \wedge \psi) \subseteq K - \phi$.

There is a close relationship between belief revision and belief contraction. That is, they can be defined by each other via the *Levi Identity* and the *Harper Identity*:

$$K * \phi = K - \neg\phi) + \phi \quad \text{(Levi Identity)}$$
$$K - \phi = (K * \neg\phi) \cap K \quad \text{(Harper Identity)}$$

It has been shown that for every revision operator $*$ satisfying AGM postulates for belief revision there is a contraction function $-$ satisfying AGM postulates for belief contraction that produces $*$ by means of the (Levi Identity). Similar comment can be applied to the Harper Identity.

AGM postulates provide nice characterization of a rational belief revision operator and a rational belief contraction operator. To justify their postulates, they also propose two constructive models for contraction operator (we only need to consider constructive models for contraction operator because of the (Levi Identity)). The first constructive model is based on a *selection function*. Contraction operators defined by selection functions are called *partial meet contraction operators*. They have shown a representation theorem which says that a partial meet contraction function corresponds the postulates $(K-1)$-$(K-8)$. The second constructive model is based on the notion of *epistemic entrenchment*, which is a special preorder on formulas representing the relative epistemic loss produced by the removal of beliefs from the agent's initial belief set. It has been shown that the class of contraction operators produced by epistemic entrenchment coincides precisely with those satisfying the AGM postulates for contraction.

Although AGM's theory on belief revision and belief contraction looks elegant, their work has been criticized by many other researchers. For example, they follow the principle of syntax independence, which is not always advisable because an agent may not want to change the syntactic form of formulas in the revised knowledge base. Another criticism is that their work is based on the assumption that an agent's beliefs should be modelled as a belief set. This representation suffers from several problems. For example, there is potentially infinite number of formulas in a belief set. Therefore, several researchers have proposed to use a belief base which is a set of formulas that is no closed under logical consequence to represent the beliefs of an agent [17,18].

In [18], AGM postulates for revision are rephrased in propositional logic as follows, where $\circ$ is a revision operator which is a function from a pair of formulas ψ and μ to a new formula denoted by $\psi\circ\mu$ (we use $\vdash$ to denote the classical deduction and $\equiv$ to denote the equivalence relation).

(R1) $\psi \circ \mu \vdash \mu$
(R2) If $\psi \wedge \mu$ is satisfiable then $\psi \circ \mu \equiv \psi \wedge \mu$
(R3) If μ is satisfiable then $\psi \circ \mu$ is also satisfiable
(R4) If $\psi_1 \equiv \psi_2$ and $\mu_1 \equiv \mu_2$ then $\psi_1 \circ \mu_1 \equiv \psi_2 \circ \mu_2$
(R5) $(\psi \circ \mu) \wedge \phi$ implies $\psi \circ (\mu \wedge \phi)$
(R6) If $(\psi \circ \mu) \wedge \phi$ is satisfiable then $\psi \circ (\mu \wedge \phi)$ implies $(\psi \circ \mu) \wedge \phi$

It has been show that, given a belief set K, if ψ is a formula that satisfies $K = Cn(\psi)$ and define $K * \mu = Cn(\psi \circ \mu)$, then $*$ satisfies $(K*1)$-$(K*8)$ if and only if $\circ$ satisfies (R1)-(R6).

Hansson in [15] defines a contraction operator, called a *kernel contraction operator*, which follows AGM's work on *safe contraction* (see [17]). He shows that a contraction operator for a belief set which is *generated by a kernel contraction* on a finite base for the belief set satisfies $(K-1)$-$(K-5)$ and some other new postulates but it does not satisfy $(K-6)$ in general. Let B be a belief base and ϕ a formula in L. A *ϕ-kernel* of B is a subset B' of B such that $B' \vdash \phi$ and there is no proper subset of B' entails ϕ. We denote the set of all ϕ-kernels by $B \perp \phi$. An *incision function* σ for B is a function such that for all ϕ, $\sigma(B \perp \phi) \subseteq \cup_{A\in B\perp\phi} A$ and for all $B' \in B \perp \phi$ such that $B' \neq \emptyset$,

$\sigma(B \perp \phi) \cap B' \neq \emptyset$. Given an incision function σ for a belief base B, we can define a kernel revision operator $-$ as follows: $B - \phi = B \setminus \sigma(B \perp \phi)$. Hansson shows that a kernel contraction operator can be characterized by a set of postulates.

Theorem 1. *[15] An operator $-$ from $2^L \times L$ to 2^L, where 2^L is the power set of L, is a kernel contraction operator if and only if it satisfies the following conditions:*

($B - 1$) If $Cn(\phi) \neq Cn(\top)$ then $\phi \notin Cn(B - \phi)$.
($B - 2$) $B - \phi \subseteq B$.
($B - 3$) If for all subsets B' of B, $\phi \in B'$ iff $\psi \in B'$, then $B - \phi = B - \psi$.
($B - 4$) If $\psi \in B$ and $\psi \notin B - \phi$, then there is a set B' such that $B' \subseteq B$ and that $\phi \notin Cn(B')$ but $\phi \in Cn(B' \cup \{\psi\})$.

To justify his postulates, Hansson gives a special kernel contraction operator, called *partial meet base contraction operator*, which is to follow AGM's partial meet construction.

To define a kernel revision operator from a kernel contraction operator, Hansson proposes the following variant of the (Levi Identity): (BL) $B * \phi = (B - \neg\phi) \cup \{\phi\}$.

Hansson and other researchers also consider another type of belief change, called *non-prioritized belief revision*, which may reject the new information. For example, Hansson in [16] proposes a *semi-revision operator*, which is defined as follows: $B \odot \phi = (B \cup \{\phi\}) - \perp$, where $-$ is a contraction function. The idea is that we first add the new information ϕ to the initial belief base B then remove all inconsistencies that may result.

4 A Survey of Revision Approaches in DLs

In this section, we present a survey of existing approaches for revision in DLs. We can roughly classify these approaches into two families: AGM-based approaches and non-AGM-based approaches. Before we introduce the approaches, we analyze why existing work on belief revision may not work in DLs and give comparison criteria.

4.1 Difficulties of Applying Belief Revisions to DLs

Although DLs are fragments of first-order logic in the sense that each axiom can be translated into a first-order formula, they have their own features which make it difficult to apply existing work on belief revision to DLs. In [21], Nebel shows that it is not possible to apply revision operators in propositional logic directly to deal with revision of the TBox of a DL knowledge base because of the following problems. First, the TBox of a DL knowledge base of some DLs can never become inconsistent, although it may be incoherent. Second, there is no counterpart of disjunction or negation of an axiom in a TBox. That is, it is impossible to state negated subsumption, like $\neg(C \sqsubseteq D)$, or disjunctive subsumption such as $(D \sqsubseteq A) \vee (B \sqsubseteq C)$ in some DL languages. It is also not possible to define disjunction between a TBox axiom and an ABox assertion in any DL. Therefore, it is not possible to define disjunction between two DL knowledge bases in most cases. It is also pointed out in [6] that a DL is not necessarily closed under the usual operators such as $\neg$ and $\wedge$. Third, there are different DL languages with different expressive power. When applying a revision operator to a particular DL, we must ensure that the result of revision can be also expressed by the same DL language. Finally, there are two kinds of contradiction in DLs: inconsistency and incoherence. Incoherence, which often occurs in the terminological level, i.e., between different TBoxes, does not provide the classical sense of the inconsistency because there might exist a model for an incoherent knowledge base. A model-based revision operator in propositional logic is usually defined by the symmetric difference between two interpretations which are a set of propositional variables. Since interpretations of a DL langauge have first-order feature, to define a model-based revision operator in DLs, we need to define the difference between two interpretations in DLs and consider if the result of revision can be expressed in the same language as the original ontology. In contrast to incoherence, an inconsistency is often caused by conflict between intensional information and extensional information, i.e., terminology axioms and assertional axioms. When dealing with inconsistency, we can either circumscribe individuals in ABoxes to restore consistency or change terminologies. One may think that we can translate a DL knowledge base to a first-order knowledge base and apply the revision operator in first-order predicate logic to it. However, this method has three weakness. First, the result of revision may be a first-order knowledge base which cannot be translated back to a DL knowledge base. Second, the advantage of using DLs is that they are decidable fragment of first-order logic, we may lose this advantage if we translate a DL knowledge base to a first-order knowledge base. Third, some DLs that provide operator like transitive closure of roles require second-order logic.

4.2 Comparison Criteria

In classical logic (propositional logic and first-order logic), two important criteria used for comparing different revision operators are postulates and computational complexity. However, these two criteria are not enough for comparison of different work on revision in DLs. First, there are several proposals of adapting existing postulates for belief revision to DLs, but there is no agreement which set of postulates is appropriate. We even need criteria for comparing different sets of postulates. Second, since one of the most important application areas of description logics is the Semantic Web, we need to consider the practical usage of the revision operators and need criteria for this purpose. To address the above two problems, we propose the following criteria (note that these criteria are what we think as important and are not claimed to be complete).

- *Implementation (IM):* Considering the vision of the Semantic Web, we should give more emphasis on the implementation of a revision approach. We consider two issues: is there an algorithm for the operator and has the algorithm been implemented?
- *Minimal Change (MC):* When we revise a knowledge base, a natural requirement is that the operator should keep as much original information as possible.
- *Preservation of Structure (PS):* We say that a revision operator preserves the structure of an axiom in an ontology if we can only remove or change some (or all) of its concept(s) or instance(s).
- *Language Dependence (LD):* Some revision operator can be applied to any DL, such as those defined in [25]. However, utilizing certain special features of DL may

lead to fine-grained revision operators and can return desirable results efficiently for certain useful languages, although these operators are not general enough for all family of DLs.

- *Uniqueness of the Result (UR):* Disjunction among axioms or ontologies is not allowed in DLs. Therefore, a revision operator either results in a single ontology or a set of ontologies. When an inconsistency or incoherence is encountered, there are usually several alternative ways to resolve it. In some applications where the user wants to participate in the revision process, they will prefer a revision operator that result in a set of ontologies. In other cases, the user may want to get a unique ontology as the result of revision.
- *Inconsistency vs. Incoherence (II):* According to the discussion in [4], incoherence is a potential cause of inconsistency but it does not provide the classical sense of the inconsistency. It is interesting to know if a revision operator can resolve incoherence or inconsistency (or both).
- *Complexity (CO):* Reasoning with expressive DLs is intractable for standard reasoning tasks. Often the approaches for dealing with inconsistencies introduce an additional level of complexity. In order to assure practicability, these complexity issues need to be taken into account.
- *Interactivity, user involvement (IU):* To revise an ontology w.r.t. another one, we usually have different alternative ways to resolve the conflict. As we have discussed before, it may be desirable to ask the user to decide which solution is the best one. Therefore, a semi-automated revision approach seems to be promising in practice.

4.3 AGM-Based Approaches

The importance of applying AGM theory to terminological systems has not been fully recognized until recent years. The first work that tries to apply AGM theory to DLs may be traced back to Nebel's thesis [21] published in 1990. He first gives five general principles for knowledge base revision in a logical language. He then argues that revision of a TBox should only be applied to definition of an atomic concept. Finally, a revision operator is defined to revise concept definitions that satisfies most of the principles. However, the five principles given by Nebel are not formalized and some of them are not well-justified. For example, he claims that a revision operator should be independent of the syntactical form of the original ontology and the newly received ontology, which is controversial. On the other hand, some of the postulates given in [6,4] and [23] follow his general principles. Furthermore, his revision operator does not resolve inconsistency and is specifically defined for DL $\mathcal{TF}$, which is not a popular DL language now. Because of these reasons and maybe other reasons, there has been no follow-up work along Nebel's theory.

Recently, there is an increasing interest in applying belief revision theory to handle inconsistency during ontology evolution. In [5,6], some work has been done to analyze the feasibility of applying AGM postulates for contraction to DLs (the underlying logic is more general than DLs in their work). Their work is based on the *coherence model*, i.e. the revised knowledge base should be knowledge set which is knowledge base closed under logical consequence (see K-1 below). They reformulate AGM postulates to a general logic. Here we restrict the logic to a DL. Suppose '$-'$' is the operation of contraction which refers to the consistent removal of a piece of information from a knowledge base. Let Cn be consequence relation in a DL language, their postulates are given as follows:

(K-1) Closure: Cn(K-X)=K-X
(K-2) Inclusion: K-X $\subseteq$Cn(K)
(K-3) Vacuity: If $X \not\subseteq$Cn(K), then K-X=Cn(K)
(K-4) Success: If $X \not\subseteq$Cn($\emptyset$), then $X \not\subseteq$Cn(K-X)
(K-5) Preservation: If Cn(X)=Cn(Y), then K-X=K-Y
(K-6) Recovery: K $\subseteq$Cn((K-X)$\cup X$)

They then define a logic to be *AGM-compliant* iff a contraction operator that satisfies the generalized AGM postulates given above can be defined in the given logic. It has been shown that some important DLs, such as $\mathcal{SHIQ}$ and $\mathcal{SHOIN}$, are not AGM compliant. They also show that by adding role operators to the $\mathcal{AL}$ family we can obtain some AGM-compliant DLs, such as $\mathcal{ALCO}^{\neg,\sqcap,\sqcup}$, and $\mathcal{ALC}^{\neg,\sqcap,\sqcup}$ with empty ABox.

It is argued in [24] that the recovery postulate for contraction is not intuitive according to the discussion in the literature and the authors introduce another postulate, called *relevance*, which is originally defined in [13]:

Relevance: If $\psi \in K$ and $\psi \notin K$-ϕ, then there is a set K' such that K-$\phi \subseteq K' \subseteq K$ and that $\phi \notin$Cn(K'), but $\phi \in$Cn($K' \cup \{\psi\}$).

They have shown that there exists a contraction operator for DL $\mathcal{SHIF}(\mathcal{D})$ and DL $\mathcal{SHOIN}(\mathcal{D})$ that satisfies generalized AGM postulates with Relevance instead of Recovery.

Considering the criteria given in last subsection, we can see that for the work given in [6] and [24], criteria IM, PS, CO and IU are not applicable. Some of their postulates capture minimal change, such as (K-3) and (K-6), but no representation theorem is given to justify their postulates. It is unknown if their approaches produce unique result or not. They are dealing with inconsistency.

None of the above work considers the explicit construction of a contraction operator that satisfies their postulates, which makes the postulates not fully justified. In contrast, in AGM's work, their postulates are well justified by some concrete contraction operators. Furthermore, they did not consider the application of AGM postulates for revision in DLs. One may wonder if we can define a revision operator by a contraction operator via the Levi identity, i.e., $K \circ X = Cn((K\text{-}\neg X) \cup X)$. The problem is that Levi identity is not applicable for most DLs [6] because negation of a DL axiom is not well-defined.

In [4], the authors advocate that in the context of ontology evolution, it is more natural to differentiate the explicit knowledge and implied knowledge in an ontology. They then propose a set of postulates for contraction by dropping the assumption that a knowledge base or an ontology is closed under logical consequence. They also define a set of postulates for revision by introducing axiom negation. Let K and X be two DL knowledge bases, and $\circ$ be a revision operator, then we have the following postulates:

(O+1) $X \subseteq K \circ X$.
(O+2) If $K \cup X$ is consistent, then $K \circ X = K \cup X$.
(O+3) If X is consistent, then $K \circ X$ is also consistent.
(O+4) If $X \equiv Y$, then $K \circ X \equiv K \circ Y$.

Plus the following postulate which is defined by a contraction operator:

(O+5) $(K \circ X) \cap K \equiv K$-$\neg X$, where the negation of an axiom has two different definitions (consistency-negation and coherence-negation) which are given in [4].

(O+1) says that the newly received information must appear in the revised ontology. (O+2) is one of the postulates for minimal change. (O+3) states that the revised ontology should be consistent if the newly received one is. (O+4) is the postulate for syntax-irrelevance.

Postulates (O+1)-(O+5) give some good insights on how a rational revision operator should behave. For example, they introduce negation of an axiom to define a revision operator by a contraction operator. The authors claim that their postulates correspond to postulates for revision given in [1]. Unfortunately, their reformulation of AGM postulates deviate the original idea of AGM theory. They ignore one important assumption underlying AGM postulates for revision given in [1]: the result of revision should be a knowledge set, instead of a knowledge base. Indeed, according to (O+2), the result of revision should be a DL knowledge base which may not be a knowledge set. Furthermore, in AGM theory, most of the revision operators satisfying AGM postulates, such as the partial meet revision operator, are defined by disjunction of some knowledge bases containing the newly received formula. The reason for using disjunction is that when revising a knowledge base, there are usually several alternative ways to resolve an inconsistency, and one does not know which solution is the best one. According to (O+1) and (O+2), the result of revision must be a single ontology which contains the new ontology X.

Let us consider the comparison criteria. Again IM, PS, CO and IU are not applicable. Some of their postulates capture minimal change, such as (O+2) and (O+5), but more justifications are needed. Their postulates can be applied to all DLs extended with negation of axioms, produce a unique ontology, and can be applied to deal with both inconsistency and incoherence.

In parallel to the work in [4], Qi et.al. in [23] propose a set of postulates for knowledge base revision in DLs based on reformulated AGM postulates given in [18]. They define a revision operator as a mapping from a pair of DL knowledge bases to a *disjunctive DL knowledge base*. A *disjunctive DL knowledge base* $\mathcal{K}$ (or DKB), defined in [20], is a set of DL knowledge bases. An interpretation is a model of a disjunctive DL knowledge base $\mathcal{K}$ iff it is a model of one of the DL knowledge bases in $\mathcal{K}$. Given a knowledge base K (resp. a disjunctive knowledge base $\mathcal{K}$), we use $M(K)$ (resp. $M(\mathcal{K})$) to denote the set of all its models. $\mathcal{K} \models K$ if and only if $M(\mathcal{K}) \subseteq M(K)$. Let K and K' be DL knowledge bases. We introduce their postulates as follows:

(G1) $M(K \circ K') \subseteq M(\phi)$ for all $\phi \in K'$
(G2) If $M(K) \cap M(K') \neq \emptyset$, then $M(K \circ K') = M(K) \cap M(K')$
(G3) If K' is consistent, then $M(K \circ K') \neq \emptyset$
(G4) If $M(K) = M(K_1)$ and $M(K') = M(K_2)$, then $M(K \circ K') = M(K_1 \circ K_2)$.
(G5) $M(K \circ K') \cap M(K'') \subseteq M(K \circ (K' \cup K''))$
(G6) If $M(K \circ K') \cap M(K'')$ is not empty, then $M(K \circ (K' \cup K'')) \subseteq M(K \circ K') \cap M(K'')$

(G1) guarantees that the new information is inferred from each revised knowledge base. (G2) requires that when there is no conflict between K and K', we do not need to change the original knowledge base. (G3) is a condition preventing a revision from introducing unwarranted inconsistency. (G1)-(G3) corresponds to (O+1)-(O+3). (G4) requires that the revision operator is independent of the syntactical forms of both the original DL knowledge base and the newly received one. It is stronger than O+4. However, this postulate is not always advisable because we may want to keep the structure of DL axioms after revision. In [23], a weaker version of (G4) is proposed:

(G4)$'$ If K_1 and K_2 are element-equivalent and $M(K_1') = M(K_2')$, then $M(K_1 \circ K_1') = M(K_2 \circ K_2')$, where K_1 and K_2 are said to be element-equivalent iff there is a bijectin f from K_1 to K_2 such that for every ϕ in K_1, $M(f(\phi)) = M(\phi)$.

Similar to the work in [18], we can show a representation theorem for the postulates (G1)-(G6) or (G1)-(G3), G(4)$'$, (G5) and (G6). We first define the notion of a *faithful assignment*.

Definition 1. *Let K and K' be DL knowledge bases and Ω the set of all interpretations in the considered DL language, a total pre-order $\preceq_K$ on Ω, associated with K, is said to be a faithful assignment if the following conditions hold:*

1. *if $\mathcal{I}, \mathcal{I}' \models K$, then $\mathcal{I} \prec_K \mathcal{I}'$ does not hold.*
2. *if $\mathcal{I} \models K$ and $\mathcal{I}' \not\models K$, then $\mathcal{I} \prec_K \mathcal{I}'$ holds.*
3. *if $K \equiv K'$, then $\preceq_K = \preceq_{K'}$.*

Furthermore, $\preceq_K$ is said to be a weak faithful assignment if it satisfies condition 1 and condition 2 above and the following condition: if K and K' are element-equivalent, then $\preceq_K = \preceq_{K'}$.

The following theorem establishes the correspondence between the set of postulates and the (weak) faithful assignment.

Theorem 2. *A revision operator $\circ$ satisfies the postulates (G1)-(G6) (resp. (G1)-(G3), G(4)$'$, (G5) and (G6)) iff for any DL knowledge base K, there exists a faithful assignment (resp. weak faithful assignment) $\preceq_K$ such that $M(K \circ K') = min(M(K'), \preceq_K)$.*

Two concrete revision operators were proposed in [23], both satisfying (G1)-(G3), (G4)$'$, (G5) and (G6). When revising an ontology by another ontology, they propose to circumscribe minimal number of individual that are responsible for the conflict from axioms in the TBox and/or minimal number of axioms from the ABox in the original ontology to restore consistency. Therefore, their revision operators are more fine-grained than those resolve inconsistency by deleting the whole axioms. One weakness of their work is that their revision operators deal with only inconsistencies arising due to objects being explicitly introduced in the ABoxes.

With respect to comparison criteria, their operators are not implemented and does not allow user involvement. According to Theorem 2, their postulates capture minimal change. The revision operators change the structure of axioms which are involved in the conflict, but keep others intact. The postulates can be applied to all DLs, but their revision operators can be only applied DLs with nominals. It is clear that the operators do not result in a unique ontology, and deal with inconsistency only.

4.4 Non-AGM-Based Approach

In [12], Hansson's semi-revision operator is adapted to DLs and an algorithm is given to implement the revision operator. However, a semi-revision operator does not guarantee that the newly received information will be kept after revision (i.e., (R1) in [4] or (G1) in [23]). Therefore, in [25], two revised semi-revision operators are given to guarantee this property. Unlike the approach in [23], their approach deletes the whole axiom in ABox and TBox to restore consistency. One of their revision operator, called *kernel revision without negation* can be equivalently defined by some postulates. Let K be an ontology, ϕ and ψ be axioms. Then $\circ$ is a kernel revision operator without negation iff it satisfies the following postulates:

[success] $\phi \in K \circ \phi$
[weak consistency] If ϕ is consistent, then $K \circ \phi$ is consistent
[inclusion] $K \circ \phi \subseteq K \cup \{\phi\}$
[core-retainment] If $\psi \in K$ and $\psi \notin K \circ \phi$ then there is at least a consistent subset K' of $K \cup \{\phi\}$ such that $K \cup \{\psi\}$ is inconsistent.
[pre-expansion] $(K \cup \{\phi\}) \circ \phi = K \circ \phi$

The result of the kernel revision operator is a single ontology according to postulate [success]. This postulate corresponds to postulate (O+1). Together with postulate [success], postulate [inclusion] says that the revision operator takes the subset of the original ontology and add the new axiom ϕ to it. Postulate [core-retainment] states that if an axiom is deleted after revision, then this axiom must belong to a sub-ontology that is in conflict with the new axiom. By [inclusion] and [core-retainment], we can infer that the kernel revision operator also satisfies the following postulate which is similar to [O+2]: If $K \cup \{\phi\}$ is consistent, then $K \circ \phi = K \cup \{\phi\}$.

The analysis of the postulates given in [25] is similar to analysis of postulates given in [4], so we do not provide it here.

The work in [10] describes a process to support the consistent evolution of OWL DL based ontologies, which ensures that the consistency of an ontology is preserved when changes are applied to the ontology. Two algorithms were given to find *minimal inconsistent sub-ontologies* and *maximal consistent sub-ontologies*. There is no guarantee that their algorithms can find all the minimal inconsistent sub-ontologies or maximal consistent sub-ontologies, although they can efficiently find one minimal inconsistent sub-ontology and one maximal consistent sub-ontology. They consider only adding one axiom (instead of an ontology) to an ontology consistently. Their algorithms have been implemented. It is clear that the revision approach for finding one maximal consistent sub-ontology captures the principle of minimal change. Since the approach only takes one maximal consistent sub-ontology, it preserves the structure of every axiom, can be applied to any DLs, and produces a unique ontology. But it only deals with inconsistency and has no user interactivity.

4.5 Comparison

The results of comparison are compactly summarized in Table 1. We do not list the approaches in [21], in [24], and in [12]. The approach in [24] is similar to the approach 1 in [6], so the result in Table 1 can be applied to it as well. The approach in [12] is not a

Table 1. Approach 1:approach in [6], Approach 2:approach in [4], Approach 3a: postulates in [23], Approach 3b: revision operators in [23], Approach 4: approach in [25], Approach 5: maximal consistent subset-based approach in [10], n.a. means "not applicable". DLs($\mathcal{O}$) in low 5 means DLs with nominals, and IS in row 7 means "inconsistency".

Criteria	Approach 1	Approach 2	Approach 3a	Approach 3b	Approach 4	Approach 5
IM	n.a.	n.a.	n.a.	no	no	yes
MC	yes	yes	yes	yes	yes	yes
PS	n.a.	n.a.	n.a.	partially	yes	yes
LD	see [6]	DLs with negation of axioms	all DLs	DLs($\mathcal{O}$)	all DLs	all DLs
UR	unknown	yes	no	no	yes	yes
II	IS	both	IS	IS	IS	IS
IU	n.a.	n.a.	n.a.	no	no	no

revision operator in a strict sense because it does not satisfy the success postulate, which is a mandatory postulate for a revision operator. Furthermore, this approach is similar to approach 4 in [25]. We also do not discuss complexity of different approaches because most of them do not have an algorithm for implementation.

According to Table 1, the syntax-based revision approaches (i.e. approach 4 and approach 5) seem to be more promising than other approaches. These approaches can be very easily generalized to deal with incoherence as well. For example, it is possible to apply the approaches for repairing terminologies in [22,26] to revise terminologies. However, these approaches are dependent on the syntactically form of the axioms and are not fine-grained when repairing inconsistency. The revision operators given in [23] are fine-grained, but they can only be applied to DLs with nominals and there is no algorithm to implement them. None of the revision operators allow user to decide how to change the ontology to restore consistency. For postulates, both sets of postulates given in [6] and [4] can only be applied to some particular DLs. In contrast, the sets of postulates given in [23] and [25] can be applied to all DLs. The only approach that considers incoherence is the one given in [4]. Surprisingly, although revision in DLs has attracted much attention, only the approach given in [10] has been implemented.

5 Discussion

In this section, we try to answer the questions given in Section 1. We also briefly discuss the relationship between revision and update in DLs.

For the first question, we have the following definition of a revision operator. Let $\mathcal{DKB}_L$ denote the set of all disjunctive DL knowledge base in a DL language L. A revision operator $\circ$ in a DL language L is a function from $2^L \times 2^L$ to $\mathcal{DKB}_L$ such that $K \circ K' \models K'$, for any DL knowledge bases K and K'. This definition has the following requirements for a revision operator in DLs: (1) the result of revision is a disjunctive DL knowledge base, (2) each ontology obtained by revision should be expressed by the same language and (3) every axiom in the newly received ontology can be inferred from the result of revision. According to our discussion before, these conditions are prerequisite for a revision operator. Note that condition (1) does not disallow the result of revision to be a single ontology, i.e. if $K \circ K'$ is a singleton then the result of revision can be equivalently considered as a single ontology due to the semantics of a disjunctive DL knowledge base. Similar to the definition of a revision operator in classical logic, our definition of a revision operator is very general. For example, $K \circ K'$ may contain inconsistent ontologies. To obtain a rational revision operator that fulfils the principle of minimal change, we need to further constrain the revision operator by some logical postulates. This will be discussed in the following.

For the second question, there is no unique answer to it because there are different sets of postulates for a revision operator in DLs. The sets of postulates given in [23] and [25] capture the principle of minimal change. Therefore, they can be used to define a rational revision operator. The difference between the postulates in [23] and those in [25] is that the former characterizes those revision operators defined by a total pre-order whilst the latter characterizes those revision operators defined by an incision function which selects axioms to be deleted. Therefore, the former allows more fine-grained approaches for resolving inconsistency. In contrast, the sets of postulates given in [6] and [4] lack of justifications for minimal change and are hard to be used.

There are a couple of works discussing update in DLs, such as [19,9]. There have been many discussions of the relationship between belief revision and belief update in the literature (see, for example, Section 8.8 of Chapter 8 in [28]). These discussions can be also applied to revision and update in DLs. For example, one of the main differences between revision and update is that when two ontologies are not in conflict, a rational revision operator will simply take their union as the result of revision to ensure minimal change, whilst a rational update operator may still change the original ontology.

6 Conclusion and Open Problems

In this paper, we presented a brief review of existing approaches for revision in DL-based ontologies. Focusing on analyzing these methods by looking into their origins in belief revision theory, we aim to clarify the pros and cons of each method. A set of criteria was proposed to compare them. The results of comparison were compactly summarized in Table 1. Based on our analysis, we gave our definition of a revision operator in DLs and our view of postulates for a revision operator. Revision in DLs is a challenging problem and need more effort to solve. We list some of the open problems for further study. First, it would be interesting to know how postulates given in [6] and [4] ensure minimal change. Second, it is also interesting to implement a revision tool which allows the user to make some decision during revision process. Third, most of work on revision in DLs considers resolving inconsistency. However, incoherence is also a defect in an ontology. Therefore, it would be interesting to discuss postulates for a revision operator that deals with incoherence.

References

1. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. J. Symb. Log. 50(2), 510–530 (1985)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, implementation and application. Cambridge University Press, Cambridge (2003)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284(5), 3443 (2001)

4. Flouris, G., Huang, Z., Pan, J.Z., Plexousakis, D., Wache, H.: Inconsistencies, negations and changes in ontologies. In: Proc. of AAAI 2006, pp. 1295–1300 (2006)
5. Flouris, G., Plexousakis, D., Antoniou, G.: Generalizing the agm postulates: preliminary results and applications. In: Proc. of NMR 2004, pp. 171–179 (2004)
6. Flouris, G., Plexousakis, D., Antoniou, G.: On applying the AGM theory to DLs and OWL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 216–231. Springer, Heidelberg (2005)
7. Fuhrmann, A.: Theory contraction through base contraction. Journal of Philosophical Logic 20(2), 175–203 (1991)
8. Gärdenfors, P.: Knowledge in Flux-Modeling the Dynamic of Epistemic States. The MIT Press, Cambridge (1988)
9. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proc. of AAAI 2006 (2006)
10. Haase, P., Stojanovic, L.: Consistent evolution of owl ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 182–197. Springer, Heidelberg (2005)
11. Haase, 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)
12. Halaschek-Wiener, C., Katz, Y., Parsia, B.: Belief base revision for expressive description logics. In: Proc. of OWL-ED 2006 (2006)
13. Hansson, S.O.: New operators for theory change. Theoria 55, 114–132 (1989)
14. Hansson, S.O.: Reversing the levi identity. Journal of Philosophical Logic 22(6), 637–669 (1993)
15. Hansson, S.O.: Kernel contraction. J. Symb. Log. 59(3), 845–859 (1994)
16. Hansson, S.O.: Semi-revision (invited paper). Journal of Applied Non-Classical Logics 7(2), 151–175 (1997)
17. Hansson, S.O.: A Textbook of Belief Dynamics: Theory Change and Database Updating. Kluwer Academic Publishers, Dordrecht (1999)
18. Katsuno, H., Mendelzon, A.O.: Propositional knowledge base revision and minimal change. Artif. Intell. 52(3), 263–294 (1992)
19. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Updating description logic aboxes. In: Proc. of KR 2006, pp. 46–56 (2006)
20. Meyer, T., Lee, K., Booth, R.: Knowledge integration for description logics. In: Proc. of AAAI 2005, pp. 645–650 (2005)
21. Nebel, B.: Reasoning and Revision in Hybrid Representation Systems. LNCS, vol. 422. Springer, Heidelberg (1990)
22. Plessers, P., De Troyer, O.: Resolving inconsistencies in evolving ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 200–214. Springer, Heidelberg (2006)
23. Qi, G., Liu, W., Bell, D.A.: Knowledge base revision in description logics. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 386–398. Springer, Heidelberg (2006)
24. Ribeiro, M.M., Wassermann, R.: First steps towards revising ontologies. In: Proc. of WONTO 2006 (2006)
25. Ribeiro, M.M., Wassermann, R.: Base revision in description logics - preliminary results. In: Proc. of IWOD 2007, pp. 69–82 (2007)
26. Schlobach, S., Huang, Z., Cornet, R., van Harmelen, F.: Debugging incoherent terminologies. J. Autom. Reasoning 39(3), 317–349 (2007)
27. Stojanovic, L.: Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe (2004)
28. van Harmelen, F., Lifschitz, V., Porter, B.: Handbook of Knowledge Representation. Elsevier Science, Amsterdam (2007)

Towards Rule-Based Minimization of RDF Graphs under Constraints

Michael Meier*

University of Freiburg, Institute for Computer Science
Georges-Köhler-Allee, 79110 Freiburg, Germany
meierm@informatik.uni-freiburg.de

Abstract. The Resource Description Framework (RDF) is a cornerstone of the Semantic Web. Due to its few and elementary language constructs, RDF data can become large and contain redundant information. So far, techniques for eliminating redundancy rely on the generic notion of lean graphs. We propose a user-specific minimization technique based on Datalog rules, enabling a user to specify the structures in an RDF graph that are not relevant for an application and therefore are deleted, while still by means of the rules retaining the possibility to reconstruct the deleted data. We set this scenario on top of constraints to ensure data consistency, i.e. if an RDF graph satisfies some constraints before minimization, these constraints must be also satisfied afterwards. The problem is decidable but already for a restricted case intractable. In addition we give a fragment of the minimization problem which can be solved in polynomial time.

1 Introduction

The Semantic Web [4] facilitates semantic interoperability and data exchange between applications. The Resource Description Framework [18] was proposed by the World Wide Web consortium as a standard language for data in the Semantic Web. As RDF has only very simple language constructs, RDF data often becomes large. There has been a line of research [8,11,12,13,14] to minimize RDF graphs without losing any information, i.e. retaining homomorphic equivalence. This allows applications to exchange reduced data, thus minimizing storage cost, transfer and query evaluation time. In [11,12,13] the notion of lean graphs was introduced as a minimal representation of an RDF graph. Basically, a lean graph eliminates triples which contain blank nodes that specify redundant information. For example in the graph $\{(a_1, a_2, a_3), (X, a_2, Y)\}$ the triple (X, a_2, Y) can be eliminated (X, Y are blank nodes) because both X and Y are treated like existentially quantified variables in the RDF semantics [13] and the triple (a_1, a_2, a_3) is a witness for the existence of such a resource (X, a_2, Y). In [8,14] different algorithms are introduced that approximately compute a lean

* The work of the author was funded by Deutsche Forschungsgemeinschaft, grant GRK 806/2.

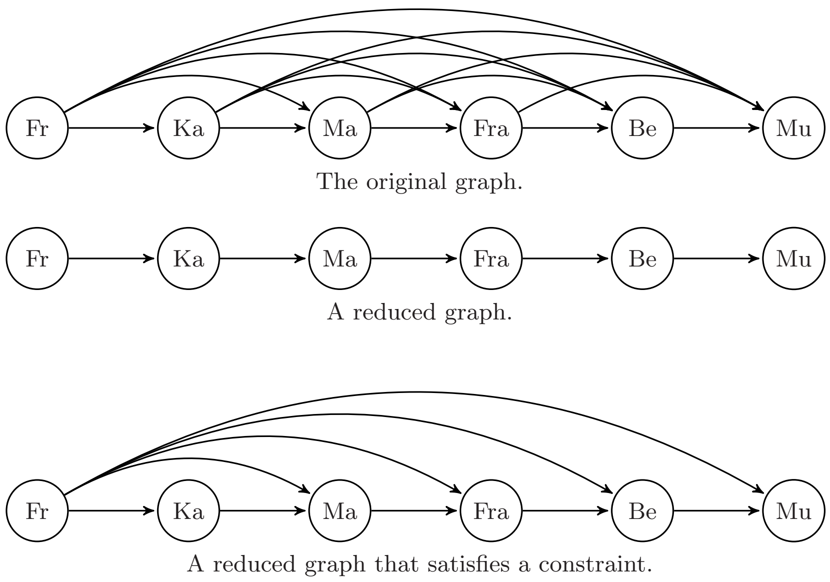


Fig. 1. An RDF graph modelling some train connections

version of a given RDF graph. The notion of lean is orthogonal to derivability by application-specific rules. If such rules exist, a lean graph may still contain triples which are redundant in the sense that they need not be stored explicitly because they could be derived by the rules as well. We propose a user-specific redundancy elimination technique based on rules. Before describing it, we give an example for our scenario. Consider figure 1. It shows the original RDF graph that models some train connections between cities. If the transitive edges are not relevant for an application, we may want to eliminate them as shown in the second graph in the figure[1]. Yet, if an application often asks for connections from 'Fr' to any other city, then we want to keep all outgoing edges from 'Fr' in order to avoid unnecessary recomputations. This is depicted in the third graph. What we have done is the following. First, we eliminated triples according to the rule "delete all transitive edges" and second we satisfied the constraint "keep all outgoing edges from 'Fr'". The constraint expresses that we are mainly interested in connections from 'Fr' to the other cities. If we would be looking for connections from 'Ka' to 'Mu' we would have to perform some additional computations on the reduced graph.

Usually, rules are interpreted generatively, i.e. if we have a rule of the form $p(X, Y) \leftarrow r(Y, X)$ and we find $r(a, b)$ in our data, we add $p(b, a)$ to it. In our work, we use rules in the following sense: whenever $p(b, a)$ and $r(a, b)$ are in our data, we delete $p(b, a)$. If later needed, we can recompute the tuple $p(b, a)$ again with the help of our rule, i.e. we minimize a given RDF graph such that all

[1] An application of transitively reduced graphs was given for the context of transitive reduction [2] in [21].

deleted triples can be reconstructed. We want to stress that our work is well-suited for scenarios in which it is known in advance what structures in the RDF graph are redundant and therefore building appropriate rules is possible. For example, this applies when the graph to be minimized has a user-defined rule semantics in the sense of [20], where it could be desirable to minimize a given graph along its rule semantics, for example the RDFS rules [17].

Additionally, we take data consistency into account. In [16] it was proposed to extend RDF and RDFS [17] by constraints. We adapt the notion of tuple-generating dependencies to the RDF scenario and ensure that if such a kind of constraint is satisfied before the minimization step, then it is also satisfied afterwards. We will exploit these constraints for answering conjunctive queries on the reduced graph. For certain queries it is possible to use only the reduced graph to compute the answer to a query posed over the original graph. Of course, this is not always possible and for another case we show that we can guarantee a non-empty answer for a query on the reduced graph if the same query yields a non-empty answer on the original graph.

The paper is structured as follows. Section 2 introduces the necessary foundations from database theory that are used throughout the paper. Section 3 introduces the minimization problem formally and gives some examples. The complexity results in section 4 show that the decision problem introduced in the previous section is intractable, namely that it is NP-complete. This is why we give a tractable fragment of it in section 5. In section 6 we consider the problem of answering queries on the reduced graph. Section 7 contains related work and finally section 8 concludes the paper.

2 Preliminaries

Throughout the paper, we will use several results from database theory that we adapt to the case of RDF. We introduce them in this section.

2.1 General Mathematical Notation

For sets M and N, $M \subset N$ denotes that M is a proper subset of N. If M is finite, then $|M|$ is the cardinality of M. We use the convention that the natural numbers $\mathbb{N}$ begin with 1 and not with zero. For $n \in \mathbb{N}$, $[n] := \{1, ..., n\}$. For a mapping f, we denote by $f \upharpoonright_M$ the restriction of f to M, where M is a subset of f's domain.

2.2 Syntax of RDF

The Semantic Web necessitates data in a machine-readable format. RDF databases are sets of triples (s, p, o). Such a triple states a directed relationship p between s and o. More formally, an RDF vocabulary is a triple (U, B, L), where U, B, L are infinite sets. U is usually referred to as the set of URI references, B is the set of blank nodes and L the set of literals. An RDF graph

G (with respect to (U, B, L)) is a finite subset of $(U \cup B) \times U \times (U \cup B \cup L)$. RDF graphs have a graphical representation. A triple $(s, p, o) \in G$ can be seen as two nodes connected by a labelled arc $s \xrightarrow{p} o$. In a triple (s, p, o), s is called the subject, p the predicate and o the object of the triple. The subset of U that occurs in an RDF graph G is denoted by U_G. B_G and L_G are defined similarly. A map (with respect to (U, B, L)) is a function $\nu : (U \cup B \cup L) \rightarrow (U \cup B \cup L)$ such that for all $x \in (U \cup L) : \nu(x) = x$. For matters of convenience we introduce the following notion. A maplet[2] (with respect to (U, B, L) and G) is a function $\mu : (U \cup B_G \cup L) \rightarrow (U \cup B \cup L)$ such that for all $x \in (U \cup L) : \mu(x) = x$. If the image of such a μ is contained in a graph G', we will denote this by $\mu : G \rightarrow G'$. Two RDF graphs G_1, G_2 are called homomorphically equivalent if there are maps μ_1, μ_2 such that $\mu_1(G_1) \subseteq G_2$ and $\mu_2(G_2) \subseteq G_1$.

2.3 Constraints in RDF

Logical constraints are a useful tool to help modelling an application domain. They restrict the legal state space of a database and guarantee that only meaningful data can be inserted into a database. In [16] it was proposed to add constraints to RDF in order to ensure data consistency. We are going to introduce a large class of constraints for RDF graphs, namely we will adapt the notion of tuple-generating dependencies.

Definition 1. *Let $m, n \in \mathbb{N}$, $x_1, ..., x_n, y_1, ..., y_m \in B$ and G, G' non-empty RDF graphs such that $B_G \subseteq \{x_1, ..., x_n\}$ and $B_{G'} \subseteq \{x_1, ..., x_n, y_1, ..., y_m\}$. A tuple generating dependency (TGD) φ is an expression of the form $\forall x_1 ... \forall x_n (G \rightarrow \exists y_1 ... \exists y_m G')$. Given $b \in \mathbb{N}$, φ is called b-bounded if $|G| \leq b$ and $|G'| \leq b$. A set $\mathcal{C}$ of TGDs is called b-bounded if every element of $\mathcal{C}$ is b-bounded. A TGD is called full dependency if it does not contain existential quantifiers.*

Note that a blank node can never occur as a predicate of an RDF triple. The only sort of constraint introduced in [16] that can be compared to TGDs are participation constraints. Clearly, TGDs generalize them. A TGD is interpreted as a semantic constraint in the following sense.

Definition 2. *An RDF graph G (with respect to (U, B, L)) satisfies a TGD $\varphi := \forall x_1 ... \forall x_n (G_1 \rightarrow \exists y_1 ... \exists y_m G_2) \Leftrightarrow$ for every maplet $\mu : G_1 \rightarrow G$ there exists a maplet $\nu : G_2 \rightarrow G$ such that for all $x \in B_{G_1} \cap B_{G_2} : \mu(x) = \nu(x)$. In the case that G satisfies φ, we write $G \models \varphi$. If $\mathcal{C}$ is a set of TGDs, we write $G \models \mathcal{C}$ if G satisfies every element in $\mathcal{C}$.*

Note that we did not define the semantics of a constraint in terms of interpretations of RDF graphs, but only gave an algebraic version of satisfaction. This is due to limited space in this paper. The following proposition states that checking constraints is tractable and that constraints are preserved under maplets. The proof follows from classical results , see [10,15].

[2] The notion of maplet is introduced because in many situations we are not interested in the image of blank nodes that do not occur in an RDF graph. It can always be extended to a map.

Proposition 3.

1. *Let $b \in \mathbb{N}$ fixed. For any RDF graph G and b-bounded set of TGDs $\mathcal{C}$, it can be tested in polynomial time whether $G \models \mathcal{C}$ holds.*
2. *Let $G \models \mathcal{C}$, where $\mathcal{C}$ is a set of TGDs, then for any maplet h it holds that $h(G) \models \mathcal{C}$ (if $h(G)$ is defined).*

2.4 The Chase

The chase is a well-known algorithm in database theory. Assume that G is an RDF graph and $\mathcal{C}$ a set of TGDs. The chase proceeds as follows. Set $G_0^{\mathcal{C}} := G$. $G_{i+1}^{\mathcal{C}}$ is obtained from $G_i^{\mathcal{C}}$ as follows. If there is some TGD $\varphi := \forall x_1 ... \forall x_n(G_1 \rightarrow \exists y_1 ... \exists y_m G_2) \in \mathcal{C}$ such that there is a maplet μ with respect to G_1 such that $\mu(G_1) \subseteq G_i^{\mathcal{C}}$, but $G_i^{\mathcal{C}} \not\models \varphi$, then we choose some fresh blank nodes $b_1, ..., b_m$ and define a mapping $h(y_j) := b_j, j \in [m]$. In this case $G_{i+1}^{\mathcal{C}} := G_i^{\mathcal{C}} \cup h(G_2)$. If for some $G_k^{\mathcal{C}}$ it happens that this procedure cannot be applied anymore, $chase_{\mathcal{C}}(G) := G_k^{\mathcal{C}}$, otherwise $chase_{\mathcal{C}}$ is undefined. The procedure minimalistically adds consequences such that the constraints are fullfilled. Note that it is not deterministic and $chase_{\mathcal{C}}(G)$ depends on the order of the applications of the constraints. Yet, it was shown that two different chase orders lead to homomorphically equivalent results, see [9]. Therefore, we omit the order of the chase in our notation and assume that the chase was performed in some arbitrary order. If $chase_{\mathcal{C}}(G)$ is defined then $chase_{\mathcal{C}}(G) \models \mathcal{C}$. If it is undefined, it is natural to ask for conditions for the termination of the chase. A broad class of TGDs for which the chase is known to terminate for any input graph is the so-called class of weakly acyclic TGDs.

Definition 4. *[9] Let $\mathcal{C}$ be a set of TGDs. For any $x \in U \cup B \cup L$ and $p \in U$ we say that*

– *x occurs in position $p^{(1)}$ if there is some RDF triple t that occurs in $\mathcal{C}$ with $t = (x, p, ...)$ and*
– *x occurs in position $p^{(3)}$ if there is some RDF triple t that occurs in $\mathcal{C}$ with $t = (..., p, x)$.*

The dependency graph of $\mathcal{C}$ is a directed graph that has all positions as vertices and there is an edge $(p^{(i)}, q^{(j)})$ if there is some constraint of the form $\forall x_1 ... \forall x_n(G_1 \rightarrow \exists y_1 ... \exists y_m G_2) \in \mathcal{C}$ such that

1. *some x_l occurs in $p^{(i)}$ in G_1 and in $q^{(j)}$ in G_2,*
2. *some x_l occurs in $p^{(i)}$ in G_1 and some y_k occurs in $q^{(j)}$ in G_2 (in this case we label the edge as existential).*

$\mathcal{C}$ is called weakly acyclic if its dependency graph contains no cycles with an existential edge.

Note that our defintition of TGDs is just a translation to RDF of the classical notion. For example, a TGD of the form $\forall x(\{(x, d, e)\} \rightarrow \{(x, f, e)\})$ can be seen as the constraint $\forall x(d(x, e) \rightarrow f(x, e))$, where d, f become predicate symbols. This is why the following theorem also applies to our case.

Theorem 5. *[9] For any set of weakly acyclic TGDs $\mathcal{C}$, there is $b \in \mathbb{N}$ such that for any RDF graph G it holds that $chase_{\mathcal{C}}(G)$ is defined and can be computed in time $O(|G|^b)$.*

In the context of the chase we consider a special class of maplets, which preserve certain blank nodes.

Definition and Convention 6. *Let $\widetilde{G} \subseteq G$ be an RDF graph and $\mathcal{C}$ a set of TGDs. Throughout the rest of the paper, we will tacitly assume that $chase_{\mathcal{C}}(\widetilde{G})$ only introduces blank nodes via existential quantifiers that do not occur in G. Then, $Hom(chase_{\mathcal{C}}(\widetilde{G}), G) := \{ \mu : chase_{\mathcal{C}}(\widetilde{G}) \to G \mid \text{f.a. } x \in B_{\widetilde{G}} : \mu(x) = x \}$.*

Note that $Hom(chase_{\mathcal{C}}(\widetilde{G}), G)$ depends also on $\widetilde{G}$ and not only on $chase_{\mathcal{C}}(\widetilde{G})$, but for readability we will omit this dependency in our notation.

2.5 Datalog

Rules allow to derive new knowledge from given knowledge and especially add recursion to a database query language. We define syntax and semantics of Datalog.

Definition 7. *A (Datalog) rule is of the form $t \leftarrow G$, where t is an RDF triple and G is an RDF graph such that $B_{\{t\}} \subseteq B_G$. Given $b \in \mathbb{N}$, $t \leftarrow G$ is called b-bounded if $|G| \leq b$. A set of rules $\mathcal{R}$ is called b-bounded if every element of $\mathcal{R}$ is b-bounded. The set $head(\mathcal{R})$ is defined as $\{ p \mid \exists s, o : (s,p,o) \leftarrow G \in \mathcal{R} \}$.*

Definition 8. *Let G be an RDF graph. The semantics of a set of rules $\mathcal{R} = \{t_1 \leftarrow G_1, ..., t_n \leftarrow G_n\}$ is defined via the help of the $T_{\mathcal{R}}$-operator, where $T_{\mathcal{R}}(G) := G \cup \{ \mu(t_i) \mid i \in [n]$ and there is a maplet $\mu : B_{G_i}$ with $\mu(G_i) \subseteq G \}$. The semantics of $\mathcal{R}$ applied to G is $\mathcal{R}(G) := \bigcup_{i=1}^{\infty} T^i_{\mathcal{R}}(G)$.*

The $T_{\mathcal{R}}$-operator is monotonic, therefore $\mathcal{R}(G) \supseteq G$. This means that we can generate new data from old one, but never lose original data. Note that $\mathcal{R}(G)$ may not be an RDF graph again because it can happen that a literal occurs in the subject position of a triple. The following proposition states that evaluating rules takes polynomial data complexity and is well-known in database theory, see [1].

Proposition 9. *Let $b \in \mathbb{N}$ fixed. Then, for any RDF graph G and any b-bounded set of rules $\mathcal{R}$ there exists $n \in \mathbb{N}$ such that $\mathcal{R}(G) := T^n_{\mathcal{R}}(G)$. Furthermore, the mapping $(G, \mathcal{R}) \mapsto \mathcal{R}(G)$ can be computed in polynomial time.*

3 Formal Problem Statement and Examples

This section introduces the minimization problem formally. We consider two versions of it: the construction problem and the corresponding decision problem.

As already mentioned earlier, we do not use Datalog rules to generate new data. Instead, we delete it from a given RDF graph. Intuitively, we define the inverse semantics of a rule of the form $t \leftarrow G$ in such a way that, we delete t, if G is still in the graph afterwards. This means that we are interested in subsets G' of a given RDF graph G such that $\mathcal{R}(G') \supseteq G$(or equivalently $\mathcal{R}(G') = \mathcal{R}(G)$ because Datalog rules are monotonic). Unfortunately, such a G' is, in general, not unique. This motivates the following definition.

Definition 10. *Let $\mathcal{R}$ be a set of Datalog rules. The inverse semantics of G with respect to $\mathcal{R}$ is given by $\mathcal{R}^{-}(G) := \{\ G' \subseteq G \mid \mathcal{R}(G') \supseteq G$ and there is no $G'' \subseteq G'$ with $\mathcal{R}(G'') \supseteq G\ \}$. If $G' \in \mathcal{R}^{-}(G)$, we call G' a reduction of G along $\mathcal{R}$.*

Example 11. Figure 2 shows an RDF graph G and three reductions along $\mathcal{R} = \{(X, d, Z) \leftarrow \{(X, d, Y), (Y, d, Z)\}\}$. □

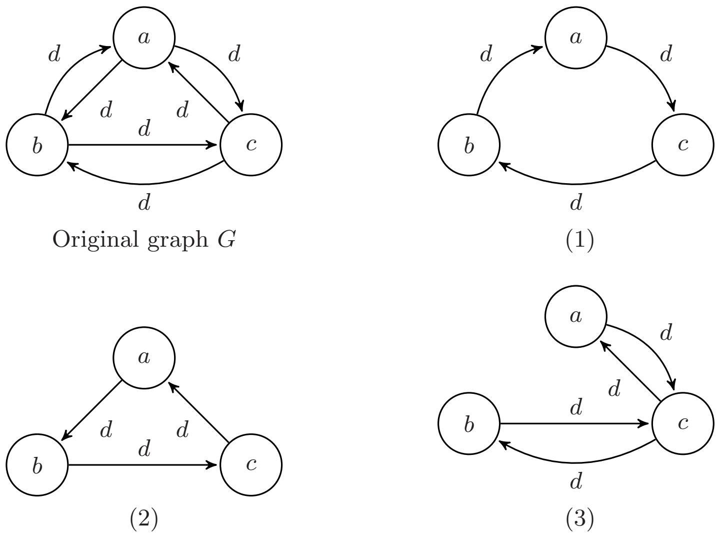


Fig. 2. An RDF graph and three possible reductions along transitivity

Definition 12. *The construction problem MINI-RDF is defined as follows.*

Input :	*An RDF graph G, a set of Datalog rules $\mathcal{R}$,*
	a set of constraints $\mathcal{C}$ with $G \models \mathcal{C}$.
Task :	*Find $G' \subset G$ such that*
	(i) $G' \models \mathcal{C}$,
	(ii) $\mathcal{R}(G') \supseteq G$ and
	(iii) G' is minimal (w.r.t. to its cardinality) with (i) and (ii).
Answer:	*G', if such a G' exists. No, otherwise.*

Note that in the definition of MINI-RDF G' must be a proper subset of G. If G' can only be chosen equal to G, then the answer to MINI-RDF will be NO. To denote that G' is a solution to MINI-RDF for the input $G, \mathcal{R}, \mathcal{C}$ we write $G' \in$ MINI-RDF$(G, \mathcal{R}, \mathcal{C})$. The corresponding decision problem MINI-RDF$_d$ is the following.

Input :	$(G, \mathcal{R}, \mathcal{C})$ *such that* $G \models \mathcal{C}$.
Question:	*Is there* $G' \in$ *MINI-RDF*$(G, \mathcal{R}, \mathcal{C})$?
Answer:	*Yes or no.*

A simple example for solutions to MINI-RDF is the following. It does not take any constraints into account.

Example 13. Consider again example 11. Let additionally be $\mathcal{C} = \emptyset$. Then, MINI-RDF$(G, \mathcal{R}, \mathcal{C})$ has the two solutions (1) and (2) depicted in figure 2. This example shows that solutions to MINI-RDF are in general not homomorphically equivalent. □

The next example makes use of a constraint. It demonstrates the interaction between the rules' inverse semantics and constraints.

Example 14. Consider again figure 1 from the introduction and assume that all edges are implicitly labelled by *reach*. The figure shows the original graph G, the only element of $\mathcal{R}^-(G)$ and the only solution to MINI-RDF$(G, \mathcal{R}, \mathcal{C})$, where
$\mathcal{R} = \{(X, reach, Z) \leftarrow \{(X, reach, Y), (Y, reach, Z)\}\}$ and
$\mathcal{C} = \{\{(Fr, reach, X), (X, reach, Y)\} \rightarrow \{(Fr, reach, Y)\}\}$. □

The following theorem gives us a method with which we are able to compute solutions to MINI-RDF using the well-known technique of the chase. The proof is given in the appendix.

Theorem 15. *If $G' \in$ MINI-RDF$(G, \mathcal{R}, \mathcal{C})$, then there exists $\widetilde{G} \in \mathcal{R}^-(G)$ and $\pi \in Hom(chase_{\mathcal{C}}(\widetilde{G}), G)$ such that $|G'| = |\pi(chase_{\mathcal{C}}(\widetilde{G}))|$.*

Given this theorem we can solve MINI-RDF in the following three steps. Of course these steps depend on each other and, in general, cannot be solved seperately:

1. Guess an adequate $\widetilde{G} \in \mathcal{R}^-(G)$.
2. Compute $chase_{\mathcal{C}}(\widetilde{G})$.
3. Find $\pi \in Hom(chase_{\mathcal{C}}(\widetilde{G}), G)$ such that $\pi(chase_{\mathcal{C}}(\widetilde{G})) \in$ MINI-RDF$(G, \mathcal{R}, \mathcal{C})$.

A natural question that arises is whether for step one it suffices to take only $\widetilde{G} \in \mathcal{R}^-(G)$ of minimal cardinality into account. The next example shows that this conjecture is wrong, in general.

Example 16. Consider the graph G from figure 2 together with the transitivity rule $\mathcal{R} = \{(X, reach, Z) \leftarrow \{(X, reach, Y), (Y, reach, Z)\}\}$ and the constraint

$\mathcal{C} = \{\{(X, d, Y)\} \leftarrow \{(Y, d, X)\}\}$. For readability we omitted all arc labels, but we assume that any arc is implicitly labelled by the URI *reach*. Then, graph (3) in the figure is a solution to MINI-RDF$(G, \mathcal{R}, \mathcal{C})$ and graphs (1) and (2) cannot be extended to a solution. Note that the graphs (1) and (2) are the only elements in $\mathcal{R}^-(G)$ of minimal cardinality. □

4 Complexity Results

This section establishes complexity results for MINI-RDF$_d$ and MINI-RDF. While it follows from the definition that the problem is decidable in general, we give an exact complexity bound for a restricted b-bounded version of MINI-RDF$_d$. Namely, we show that it is NP-complete.

Proposition 17. *Let $b \in \mathbb{N}$ fixed. MINI-RDF$_d$ restricted to instances of b-bounded sets of TGDs and b-bounded sets of rules is solvable by an NP-algorithm.*

Proof. Let $(G, \mathcal{R}, \mathcal{C})$ be an input for MINI-RDF$_d$. Non-deterministically guess $G' \subset G$ and check whether $\mathcal{R}(G') \supseteq G$ and $G' \models \mathcal{C}$. Notice that by propositions 3 and 9 these steps take polynomial time. □

Theorem 18. *MINI-RDF$_d$ restricted to instances of 2-bounded sets of full dependencies and 1-bounded sets of rules is NP-hard. This still holds if neither the rules, the constraints nor the input graph contain any blank nodes.*

The proof of theorem 18 is given in the appendix. This result demonstrates even in such a restricted case the interaction between the rules' inverse semantics and the constraints is so complex that this leads to NP-hardness.

Corollary 19. *Let $b \geq 2$ fixed. MINI-RDF$_d$ restricted to instances of b-bounded sets of TGDs and b-bounded sets of rules is NP-complete under polynomial-time many-one reductions.*

5 A Tractable Fragment

The complexity results for MINI-RDF$_d$ and therefore also for MINI-RDF from the last section are negative. We will now look for tractable subsets. The proof of theorem 18 seems to imply that recursion in the Datalog rules along with cycles in the input RDF graph are a source of high complexity. This is why we will restrict ourselves to a case where recursion is limited and the constraints are a set of full dependencies. We will give a syntactic restriction in terms of acyclicity of a graph.

Definition 20. *Let $\mathcal{R}$ be a set of rules and G an RDF graph. The data dependency graph with respect to $(\mathcal{R}, G)$ is defined as dep$(\mathcal{R}, G) = (\mathcal{R}(G), E_{\mathcal{R}})$, where $E_{\mathcal{R}} := \{\ (v, w) \mid$ there are $t \leftarrow \widetilde{G} \in \mathcal{R}, \mu : \widetilde{G} \to \mathcal{R}(G)$ such that $\mu(t) = w, v \in \mu(\widetilde{G})\ \}$. $(\mathcal{R}, G)$ is called acyclic if dep$(\mathcal{R}, G)$ is acyclic, i.e. for every node in the graph, there is no directed path to itself.*

The intuition why the acyclicity of $dep(\mathcal{R}, G)$ yields a tractable fragment of MINI-RDF is the following. We want to solve MINI-RDF according to the steps in theorem 15. The main problem in the reduction along the rules is that we do not know in what order triples should be deleted. When $dep(\mathcal{R}, G)$ is acyclic, we can impose an order on $\mathcal{R}(G)$ such that the deletion of an element that has a very high rank in this order will not affect the deletion of an element with a low rank. Deleting triples according to that order yields the unique reduction of G along $\mathcal{R}$.

Remark 21

1. *If $(\mathcal{R}, G)$ is acyclic, then $|\mathcal{R}^{-}(G)| = 1$.*
2. *If $\mathcal{C}$ is a set of full dependencies, then $|Hom(chase_{\mathcal{C}}(\widetilde{G}), G)| = 1$ for any $\widetilde{G} \subseteq G$.*

The proof of remark 21 is given in the appendix. As a consequence, we obtain the following corollary, which states that under the conditions of the remark, computing solutions to MINI-RDF is tractable if we consider data complexity[3]. A proof for the corollary is also given in the appendix.

Corollary 22. *Let $\mathcal{R}$ be a fixed set of rules and $\mathcal{C}$ be a fixed set of constraints. For any RDF graph G auch that $(\mathcal{R}, G)$ is acyclic, a solution to MINI-RDF $(G, \mathcal{R}, \mathcal{C})$ can be computed in polynomial time (with respect to $|G|$).*

Consider again $\mathcal{R}, G$ from example 14. It can be easily seen that $(\mathcal{R}, G)$ is acyclic there.

Next, we will show that a special case of $(\mathcal{R}, G)$ being acyclic is that $\mathcal{R}$ is acyclic, i.e. non-recursive. We first repeat the definition of an acyclic set of rules.

Definition 23. *Let $\mathcal{R}$ be a set of rules.*

1. *The dependency graph $dep(\mathcal{R}) = (V_{\mathcal{R}}, E_{\mathcal{R}})$ is defined as*
   $V_{\mathcal{R}} := \{\ v \mid v$ is a predicate symbol that occurs in $\mathcal{R}\ \}$ and
   $E_{\mathcal{R}} := \{\ (v, w) \in V_{\mathcal{R}} \times V_{\mathcal{R}} \mid v$ occurs in the body of a rule with head $w\ \}$.
2. *$\mathcal{R}$ is called acyclic if and only if $dep(\mathcal{R})$ is acyclic.*

The following remark shows that the tractable fragment given in theorem 22 generalizes the case when the set of rules is acyclic. We give no detailed proof for it, but just mention that a cycle in $dep(\mathcal{R}, G)$ would force a cycle in $dep(\mathcal{R})$.

Remark 24. *If $\mathcal{R}$ is an acyclic set of rules, then for any RDF graph G it holds that $(\mathcal{R}, G)$ is acyclic.*

6 Query Answering

So far, we were able to reduce the size of an RDF graph via rules. Now we want to consider the problem of answering conjunctive queries posed on the original

[3] We assume that the data, i.e. the RDF graph, is much larger than the rules and the constraints.

graph using the reduced graph only. Consider a query q. As the reduced graph G' is always a subset of the input graph G, $q(G') \subseteq q(G)$. But there are some cases where $q(G) = q(G')$ or at least $q(G) \neq \emptyset \Leftrightarrow q(G') \neq \emptyset$ holds. We will briefly introduce the basic definitions and then use the chase and backchase technique [6] in order to answer queries.

Let G' be a non-empty RDF graph and V a finite subset of B such that $V \subseteq B_{G'}$. A conjunctive query q has the form $q : B \leftarrow G'$. The set *body*(q) is defined as G'. The answer of a conjunctive query on an RDF graph G is $q(G) := \{ \mu \uparrow_V \mid \mu : G' \rightarrow G$ is a maplet $\}$. Note that our notion of conjunctive queries can be easily implemented using SPARQL [19]. For two conjunctive queries q, q' we say that q is contained in q' in the presence of $\mathcal{C}$, $q \sqsubseteq_{\mathcal{C}} q'$, if for every RDF graph G such that $G \models \mathcal{C}$ it holds that $q(G) \subseteq q'(G)$. If $q \sqsubseteq_{\mathcal{C}} q'$ and $q' \sqsubseteq_{\mathcal{C}} q$ we write $q \equiv_{\mathcal{C}} q'$. In [6] the following result is shown.

Theorem 25. *[6] Let q be a conjunctive query and $\mathcal{C}$ a set of TGDs such that chase$_{\mathcal{C}}$(body(q)) is defined. Then, there is an algorithm that outputs precisely all queries q' (up to isomorphism) such that no predicate in body(q') appears in head$(\mathcal{R})$ and $q \equiv_{\mathcal{C}} q'$.*

From this theorem we obtain as a first result the possibility to answer a query on the full graph using the query and reduced graph only. We use this algorithm, called Chase and Backchase, in order to determine if it is possible to rewrite a given query such that its body does not contain any predicates that could have been minimized by the rules. Of course, this is not always possible and stronger results are left as future work. The corollary is proven in the appendix.

Corollary 26. *Let $\mathcal{R}$ be a set of rules, $\mathcal{C}$ weakly acyclic, $G' \in$ MINI-RDF $(G, \mathcal{R}, \mathcal{C})$ and $q = V \leftarrow G_0$ a conjunctive query. The query q_0 is defined as $\emptyset \leftarrow G_0$. If there is a conjunctive query q' such that no predicate in body(q') appears in head$(\mathcal{R})$ and*

1. *$q \equiv_{\mathcal{C}} q'$, then $q(G) = q(G')$.*
2. *$q_0 \equiv_{\mathcal{C}} q'$, then $q(G) \neq \emptyset \Leftrightarrow q(G') \neq \emptyset$.*

This corollary gives us a case where we can expect non-empty answers on the reduced graph for the case that we had a non-empty answer in the original graph.

7 Related Work

Our work extends the works on transitive reduction, see [2], in graph theory. Yet, in our framework it is not only possible to eliminate transivities in graphs, but to define general rules for the reduction. Constraints on the reduced graph are also not considered in [2].

The problem of optimizing the amount of data that must be exchanged if a graph is updated and the update must be transferred to other hosts was already studied in [22] using the notion of deltas. We are not aware of any work on RDF that provides user-specific minimization techniques the way we do.

The work in [8,14] eliminates triples in the sense of lean graphs and homomorphic equivalence. Although the results that are produced are homomorphically equivalent to the original graph, they are not necessarily lean. Our work does not subsume [8,14] nor the other way round. We focus on different aspects of redundancy elimination. In our work, it must be explicitly specified via rules what a redundancy structurally looks like. For example, our method may also be suitable when the graph to be minimized has a user-defined rule semantics in the sense of [20], where it could be desirable to minimize a given graph along its rule semantics. The notion of lean is orthogonal to derivability by application-specific rules. If such rules exist, a lean graph may still contain triples which are redundant in the sense that they need not be stored explicitly because they could be derived by the rules as well. Minimization in the sense of lean graphs is always possible because this method is generic.

We also want to mention that MINI-RDF is not an abduction problem in the sense of [7]. An abduction problem is characterized as follows: if we observe C and have a rule of the form $C \leftarrow A$, then we can conclude the premise A, regardless of A follows from our fact base or not. In our scenario, we would only delete C from our fact base if afterwards A still follows from it. Knowledge assimilation in deductive databases [5] uses abduction techniques to rewrite updates in such a way that they can be performed on the extensional data only. Here it is already known in advance what data is extensional and what data is intensional. In a certain sense, in our scenario we want to compute the extensional part and delete all intensional data such that the constraints remain satisfied.

8 Conclusions

We introduced the problem of minimizing RDF graphs along rules in the presence of constraints. We showed that this problem is NP-complete even for a restricted case and gave a tractable fragment of it. As a last contribution we provided a first and simple result on answering queries posed over the original graph using only the reduced graph to answer it.

There are several directions left open for future work. First of all, we did not consider negation in the rules and except for theorem 15 no existential quantifiers in the TGDs. Also other constraints than TGDs should be investigated for our scenario e.g. the constraints studied in [16], where classical database constraints were mixed with typical constraints on ontologies. Query answering with the help of the reduced graph should be looked at more thoroughly. More expressive classes of conjunctive queries than introduced in this paper and approximate query answering should be studied. On the practical side, we plan to implement our work and evaluate the results.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Add. Weas. (1995)
2. Aho, A., Garey, M., Ullman, J.: The Transitive Reduction of a Directed Graph. SIAM Journal on Computing 1(2), 131–137 (1972)

3. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Draft of a book (January 2007), http://www.cs.princeton.edu/theory/complexity/
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
5. Decker, H.: Some Notes on Knowledge Assimilation in Deductive Databases. Transactions and Change in Logic Databases, 249–286 (1998)
6. Deutsch, A., Popa, L., Tannen, V.: Query reformulation with constraints. SIGMOD Rec. 35(1), 65–73 (2006)
7. Eiter, T., Gottlob, G., Leone, N.: Abduction from Logic Programs: Semantics and Complexity. Theor. Comput. Sci. 189(1-2), 129–177 (1997)
8. Esposito, F., Iannone, L., Palmisano, I., Redavid, D., Semeraro, G.: REDD: An Algorithm for Redundancy Detection in RDF Models. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 138–152. Springer, Heidelberg (2005)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. IBM Research Report (November 2002)
10. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
11. Gutierrez, C., Hurtado, C., Mendelzon, A.: Formal aspects of querying RDF databases. In: Proc. of First International Workshop on Semantic Web and Databases, Berlin, Germany (September 2003)
12. Gutierrez, C., Hurtado, C., Mendelzon, A.O.: Foundations of Semantic Web Databases. In: ACM Symposium on Principles of Database Systems (PODS) (2004)
13. Hayes, P.: RDF semantics W3C Recommendation 10 (February 2004), http://www.w3.org/TR/rdf-mt/
14. Iannone, L., Palmisano, I., Redavid, D.: Optimizing RDF Storage Removing Redundancies: An Algorithm. In: Ali, M., Esposito, F. (eds.) IEA/AIE 2005. LNCS (LNAI), vol. 3533, pp. 732–742. Springer, Heidelberg (2005)
15. Johnson, D.S., Klug, A.: Testing containment of conjunctive queries under functional and inclusion dependencies. Journal of Computer and System Sciences 28, 167–189 (1984)
16. Lausen, G., Meier, M., Schmidt, M.: SPARQLing constraints for RDF. In: EDBT 2008 (2008)
17. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recom., February 10 (2004), http://www.w3.org/TR/rdf-schema/
18. Resource Description Framework: Concepts and Abstract Syntax. W3C Recom., February 10 (2004), http://www.w3.org/TR/rdf-concepts/
19. SPARQL Query Language for RDF. W3C Proposed Recommendation, January 15 (2008), http://www.w3.org/TR/rdf-sparql-query/
20. ter Horst, H.J.: Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 668–684. Springer, Heidelberg (2005)
21. Dubois, V., Bothorel, C.: Transitive Reduction for Social Network Analysis and Visualization. Web Intelligence 2005, 128–131 (2005)
22. Zeginis, D., Tzitzikas, Y., Christophides, V.: On the Foundations of Computing Deltas Between RDF Models. ISWC/ASWC 2007, 637–651 (2007)

Appendix

A Proof of Theorem 15

Assume that for all $\widetilde{G} \in \mathcal{R}^{-}(G)$ and for all $\pi \in Hom(chase_{\mathcal{C}}(\widetilde{G}), G)$ it holds that $|G'| \neq |\pi(chase_{\mathcal{C}}(\widetilde{G}))|$.

Case 1: Assume that there is $\widetilde{G} \in \mathcal{R}^-(G)$ and $\pi \in Hom(chase_\mathcal{C}(\widetilde{G}), G)$ such that $|G'| > |\pi(chase_\mathcal{C}(\widetilde{G}))|$. Clearly, for every choice of $\widetilde{G}$ and π we have that $\pi(chase_\mathcal{C}(\widetilde{G})) \models \mathcal{C}$ and $\mathcal{R}(\pi(chase_\mathcal{C}(\widetilde{G}))) \supseteq G$. Thus, $G' \notin$ MINI-RDF$(G, \mathcal{R}, \mathcal{C})$.
Case 2: Assume that for all $\widetilde{G} \in \mathcal{R}^-(G)$ and for all $\pi \in Hom(chase_\mathcal{C}(\widetilde{G}), G)$ it holds that $|G'| < |\pi(chase_\mathcal{C}(\widetilde{G}))|$. Define $A := \{\ G'' \subseteq G' \mid G'' \in \mathcal{R}^-(G)\ \}$. By definition $A \neq \emptyset$. Choose $\widetilde{G} \in A$ arbitrarily. As $\widetilde{G} \subseteq G'$ $\iota : \widetilde{G} \to G'$ is a maplet, where $\iota(x) = x$ for all x in the domain of ι. $G' \models \mathcal{C}$. So, there exists a maplet $\pi : chase_\mathcal{C}(\widetilde{G}) \to G'$ with $\pi \supseteq \iota$ (see [15]), thus $\pi \in Hom(chase_\mathcal{C}(\widetilde{G}), G)$ and $|\pi(chase_\mathcal{C}(\widetilde{G}))| \leq |G'|$. By assumption $|G'| < |\pi(chase_\mathcal{C}(\widetilde{G}))|$, which yields a contradiction. □

B Proof of Theorem 18

It is well-known that $CNF{-}SAT$, the satisfiability problem for boolean formulas in conjunctive normal form, is NP-complete under polynomial-time many-one reductions, see [3]. We show that $CNF - SAT$ can be reduced to MINI-RDF$_d$. An instance α of $CNF - SAT$ is of the form

$$(x_{1,1} \vee ... \vee x_{1,k_1}) \wedge ... \wedge (x_{l,1} \vee ... \vee x_{l,k_l}),$$

where $x_{i,j}$ are literals. Without loss of generality, we assume that $(x \vee \neg x)$ is a conjunct in α for every veriable x that occurs in α. $\overline{x_{i,j}} = x_{i,j}$, if $x_{i,j}$ is a positive literal and $\overline{x_{i,j}} = p$, if $x_{i,j} = \neg p$ for a positive literal p. A possible reduction is given by $\alpha \mapsto (G, \mathcal{R}, \mathcal{C})$, where

$$\begin{aligned}
G =\ & \{\ (x_{i,j}, i, x_{i,j}) \mid i \in [l], j \in [k_i]\ \} \cup \{(d,d,d)\} \\
\mathcal{R} =\ & \{\ (x_{i,j}, i, x_{i,j}) \leftarrow (x_{i,j'}, i, x_{i,j'}) \mid i \in [l], j, j' \in [k_i], j \neq j'\ \} \\
& \cup \{\ (d,d,d) \leftarrow t \mid t \in G \backslash \{(d,d,d)\}\ \} \\
\mathcal{C} =\ & \{\ ((x_{i,j}, i, x_{i,j}) \to (x_{i',j'}, i', x_{i',j'})) \mid x_{i',j'} = x_{i,j}\ \} \\
& \cup \{\ ((x_{i,j}, i, x_{i,j}), (x_{i',j'}, i', x_{i',j'}) \to t) \mid \overline{x_{i',j'}} = x_{i,j}, t \in G\ \}.
\end{aligned}$$

Note that $\alpha \mapsto (G, \mathcal{R}, \mathcal{C})$ is computable in polynomial time and that neither G, $\mathcal{R}$ nor $\mathcal{C}$ contain any blank nodes. It remains to show that α is satisfiable if and only if MINI-RDF$(G, \mathcal{R}, \mathcal{C})$ admits a solution.

Assume that b is a satisfying assignment for α. Define $G' := \{\ (x_{i,j}, i, x_{i,j}) \mid i \in [l], j \in [k_i], b(x_{i,j}) = 1\ \}$. We will show that (i) $\mathcal{R}(G') \supseteq G$, (ii) $G' \models \mathcal{C}$ and $G' \subset G$. Obviously, $\emptyset \neq G' \subseteq G$ holds.(i): Clearly, $(d,d,d) \in \mathcal{R}(G')$ because $\emptyset \neq G'$ and for any $s \in G' \backslash \{(d,d,d)\}$ there is $(d,d,d) \leftarrow s \in \mathcal{R}$. Let $t \in G \backslash \{(d,d,d)\}$ arbitrarily. Then, there exist $i \in [l], j \in [k_i]$ such that $t = (x_{i,j}, i, x_{i,j})$. As b satisfies α, b also satisfies the i-th conjunct of α. So, there is $j' \in [k_i]$ such that $b(x_{i,j'}) = 1$. But then $(x_{i,j}, i, x_{i,j}) \leftarrow (x_{i,j'}, i, x_{i,j'}) \in \mathcal{R}$ which implies $t \in \mathcal{R}(G')$. (ii): A constraint of the form $((x_{i,j}, i, x_{i,j}) \to (x_{i',j'}, i', x_{i',j'}))$ where $x_{i',j'} = x_{i,j}$ is satisfied because $b(x_{i',j'}) = b(x_{i,j})$. Constraints of the form $((x_{i,j}, i, x_{i,j}), (x_{i',j'}, i', x_{i',j'}) \to t)$ $(\overline{x_{i',j'}} = x_{i,j}, t \in G)$ are satisfied too because otherwise $b(x_{i,j}) = b(x_{i',j'}) = 1$ although $\overline{x_{i',j'}} = x_{i,j}$. So, overall we have found a proper subset of G that satisfies (i) and (ii). This implies MINI-RDF$(G, \mathcal{R}, \mathcal{C}) \neq \emptyset$.

Assume conversely that $G' \in$ MINI-RDF$(G, \mathcal{R}, \mathcal{C})$. Note that $(d,d,d) \notin G'$. We define a satisfying assignment b for α. For $i \in [l], j \in [k_i]$, if $(x_{i,j}, i, x_{i,j}) \in G'$, then $b(x_{i,j}) := 1$. Note that b is well-defined and that it cannot occur that $b(x_{i,j}) = b(\overline{x_{i,j}}) = 1$ because $G' \models \mathcal{C}$. For all literals x that occur in α but neither x nor $\overline{x}$ was assigned a truth value, set $b(x) \in \{0,1\}$ arbitrarily. Assume b does not satisfy the i-th conjunct in α. By assumption $\mathcal{R}(G') \supseteq G$. So, by construction, there must be $j \in [k_i]$ such that $(x_{i,j}, i, x_{i,j}) \in G'$. This means $b(x_{i,j}) = 1$. Thus, b satisfies the i-th conjunct in α, which is a contradiction. □

C Proof of Remark 21

1.) Define $G' := \{\ w \in G \mid \text{for all } v \in \mathcal{R}(G) : (v,w) \notin E_{\mathcal{R}}\ \}$. We will show that $\mathcal{R}^-(G) = \{G'\}$.

Assume that $G' \notin \mathcal{R}^-(G)$. Then, there is $a \in G'$ such that $\mathcal{R}(G' \backslash \{a\}) \supseteq G$. Thus, there must be a rule $t \leftarrow \widetilde{G} \in \mathcal{R}$ and a maplet $\mu : G' \backslash \{a\} \rightarrow \mathcal{R}(G)$ such that $a = \mu(t)$ and $\mu(\widetilde{G}) \subseteq G' \backslash \{a\}$. From the construction of G' it follows that $a \notin G'$, which is a contradiction.

Conversely, assume there is some $G'' \in \mathcal{R}^-(G)$. We will show that $G'' = G'$. Assume for a short moment that $G'' \neq G'$. Case 1: there is some $a \in G'' \backslash G'$. Without loss of generality, we can assume that a is minimal with respect to the order of an arbitrary topological sorting of $\mathcal{R}(G)$ according to $E_{\mathcal{R}}$. In case that a has no predecessors in $E_{\mathcal{R}}$, then $a \in G'$ by construction. Otherwise, a must have a predecessor. It follows that $a \notin \mathcal{R}^-(G'' \backslash \{a\})$. So, there must be $b \in \mathcal{R}(G) \backslash \mathcal{R}(G'')$ such that $(b,a) \in E_{\mathcal{R}}$. This is a contradiction to the minimality of a.

Case 2: there is some $a \in G' \backslash G''$. Then, it must hold that $\mathcal{R}(G') \neq \mathcal{R}(G'')$. We show that $a \notin \mathcal{R}(G'')$. Assume that $a \in \mathcal{R}(G'')$. Then there must be a rule $t \leftarrow \widetilde{G} \in \mathcal{R}$ and a maplet $\mu : \widetilde{G} \rightarrow \mathcal{R}(G)$ such that $\mu(t) = a$. As $\widetilde{G} \neq \emptyset$, a must have a predecessor in $E_{\mathcal{R}}$. But as $a \in G'$, then by construction of G' a cannot have any predecessors in $E_{\mathcal{R}}$, which is a contradiction.

2.) $|Hom(chase_{\mathcal{C}}(\widetilde{G}), G)| = 1$ for any $\widetilde{G} \subseteq G$ holds because $\mathcal{C}$ is a set of full dependencies, the chase does not introduce any variables and therefore the identity is the only element in $Hom(chase_{\mathcal{C}}(\widetilde{G}), G)$. □

D Proof of Corollary 22

Assume that G is an RDF graph such that $(\mathcal{R}, G)$ is acyclic. We give an algorithm that computes a solution to MINI-RDF$(G, \mathcal{R}, \mathcal{C})$. Let G' be as in the proof of remark 21 and note that the mapping $G \mapsto G'$ can be computed in polynomial time. Compute $chase_{\mathcal{C}}(G')$. Due to theorem 5 this can be in time polynomial in $|G'|$. If $chase_{\mathcal{C}}(G') \subset G$, then return $chase_{\mathcal{C}}(G')$, otherwise return NO. By theorem 15 and remark 21 this algorithm works correct. We already argued that it takes time polynomial in $|G|$. □

E Proof of Corollary 26

1.) Similar to point two.

2.) If $q(G) \neq \emptyset$, then $\{\emptyset\} = q_0(G) = q'(G)$ because $G \models \mathcal{C}$. Furthermore, $q'(G) = q'(G')$ because $body(q)$ contains no predicates that could have been minimized. As $G' \models \mathcal{C}$, $q'(G') = q_0(G')$. Therefore, $q(G') \neq \emptyset$. □

Ontology Design and Reuse with Conceptual Roles[*]

Jakob Henriksson[1], Michael Pradel[1], Steffen Zschaler[1], and Jeff Z. Pan[2]

[1] Lehrstuhl Softwaretechnologie, Fakultät Informatik,
Technische Universität Dresden, Germany
{steffen.zschaler|jakob.henriksson}@tu-dresden.de,
michael@binaervarianz.de

[2] Department of Computing Science, University of Aberdeen, UK
jpan@csd.abdn.ac.uk

Abstract. Ontologies are already used in the life sciences and the Semantic Web, but are expected to be deployed in many other areas in the near future—for example, in software development. As the use of ontologies becomes commonplace, they will be constructed more frequently and also become more complex. To cope with this issue, modularization paradigms and reuse techniques must be defined for ontologies and supported by ontology languages. In this paper, we propose the use of *roles* from conceptual modeling for this purpose, and show how they can be used to define ontological reuse units and enable modularization. We present role-based ontologies as an extension of standard ontologies and define their semantics through a reduction to standard Description Logics, such that existing reasoners can be used.

1 Introduction

In conceptual modeling it has long been known that there is a fundamental distinction between different kinds of concepts: some stand on their own (e.g. *Person*), while others depend on the existence of some other concept (e.g. *Borrower*, who must be related to the borrowed item). Making this distinction explicit is favored in the role modeling community (see e.g. [17,18] and references therein), with successful applications—for example, in object-oriented programming [6]. In role modeling, concepts that can stand on their own are called *natural types*, while dependent concepts are called *role types*. Even though considered an important and fundamental conceptual differentiation, current ontology languages, such as OWL DL [11], do not support it (nor does the OWL 2 working draft [2]). The Description Logics (DLs) community—providing much of the foundations for OWL DL—is however aware of the differentiation and refers to role types as *relationship-roles* [1].[1] The DL handbook even supports the idea for the ontology development process by encouraging its readers to "distinguish independent concepts from relationship-roles" [1, p. 379].

Distinguishing different kinds of concepts is not only important for a better understanding of the modeled domain, but also for ontology reuse. This second application of

[*] This research has been co-funded by the European Commission within the 7th Framework Programme project MOST number 216691 (cf. http://most-project.eu).

[1] The DL community uses the term 'role' for binary properties. To avoid confusion with conceptual roles (relationship-roles) we will use the term *dl-role* to describe the former construct.

role types has—to the best of our knowledge—never been investigated by the ontology community. Related role types and their relationships form abstraction units that can be studied and defined on their own. Such abstraction units are traditionally called *role models*. As role models often transcend domains, they can be reused in different ontologies.

Consider, for example, the simple role model in Figure 1. It describes the role types *Producer*, *Product*, and *Consumer*, and their relationships.

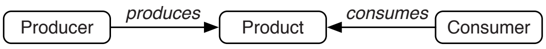


Fig. 1. A simple role model describing three related role types and their relationships

This role model could be integrated in ontologies covering topics as different as foods, wines, consumer electronics, or car dealerships. For example, a consumer electronics company modeling their infrastructure and their business processes can (re)use the generic description captured in the role model in Figure 1.

By being domain-independent units, role models have something in common with upper-level ontologies. Upper-level ontologies, however, are mainly used for integration purposes and reside "above" domain ontologies. Role models serve a different purpose, come with a different reuse approach and—as we will see—essentially reside "below" domain ontologies.

Role modeling can be seen as a design methodology, or process, allowing to break a larger description into smaller, reusable units—the role models. But to successfully deploy design processes for reuse, the underlying languages must support them. To quote Kiczales et. al. [8]: "Software design processes and programming languages exist in a mutually supporting relationship." Clearly 'programming languages' can here be substituted with 'ontology languages.' They go on to explain that the design processes allow to break a system down into smaller and smaller (reusable) units, and that "a design process and a programming language work well together when the programming language provides abstraction and composition mechanisms that cleanly support the kinds of units the design process breaks the system into." It would hence be advantageous—from a reuse point of view—if ontology languages supported the definition of role models and provided constructs for composing role models into complete ontologies.

In this paper we investigate how explicitly supporting role modeling in ontology languages can provide an important and little investigated reuse opportunity—in form of role models. We also discuss the consequences of supporting the underlying role modeling semantics. The contributions of this paper are: (1) We demonstrate the viability of role models as ontological reuse units. (2) We propose a formalization for what role modeling means for current ontology languages. (3) We explain the consequences of introducing the semantics of role modeling into ontology languages. That is, the possible reasoning effects the ontology engineer has to be aware of. (4) We describe how the role modeling semantics can be realized by reducing role modeled ontologies into ontologies expressed in standard ontology languages. This enables the reuse of reasoning engines. Preliminary results of our work have been presented in [13].

The remainder of the paper is structured as follows: We begin in the next section by giving some background on role modeling and DLs. In Section 3 we discuss an example of how role models can provide reuse opportunities, before providing a formal reduction of our concepts to an underlying DL in Section 4.

2 Background

This section gives background knowledge the rest of the paper is based on. First, we introduce role modeling as it is known from conceptual and software modeling. Then, we discuss Description Logics, the formal underpinning of many current ontology languages.

2.1 Role Modeling

In modeling, types (or concepts) abstract over sets of individuals. Role modeling at its core argues for the existence of two inherently distinguishable abstractions: *natural types* and *role types*, a terminology first introduced by Sowa [16]. Intuitively, natural types describe the part of individuals that are essential to their identity while role types describe accidental or temporal relations to other individuals.

A common example is the natural type *Person* and its associated role type *Actor*. In this case, a person is said to *play the role* of an actor. Along with being an actor comes, for example, giving performances led by stage managers and attending rehearsals led by directors. Hence, in the role of being an actor one stands in certain relations to individuals of other role types such as *Director*.

Guarino defines natural types and role types using the notions of *founded types* and *semantically rigid types* [3]. A type is founded if all of its individuals have to be related to an individual of another type, where the relation is not part-of. For example, one could say that an actor necessarily has to be related to a director in order to be an actor. A type is semantically rigid, if it contributes to the identity of its individuals. For instance, the name and date of birth of a person are part of its identity. Hence, an individual cannot drop the type *Person*, while ceasing to be an *Actor* is possible. Using these two notions, we can define natural types and role types:

– A *natural type* is non-founded and semantically rigid.
– A *role type* is founded and semantically non-rigid.

Although the notion of roles seems intuitively clear, different definitions exist in the literature. Steimann summarizes the fifteen most common characteristics the research community associates with roles in object-oriented and conceptual modeling [17]. As an ontology describes a static view of the world, those referring to dynamic aspects observable in software systems are not relevant in this paper. Among the remaining, we consider the following to be the most fundamental (each of Steimann's role features is referred to as 'Sx'):

S1 "*A role comes with its own properties and behaviour.* This basic property suggests that roles are types."
S2 "*Roles depend on relationships.* As suggested by the work of Sowa and Guarino, a role is meaningful only in the context of a relationship."

S3 "*An object may play different roles simultaneously.* This is one of the most broadly accepted properties of the role concept."

S14 "*An object and its roles share identity.* In the object-oriented world this entails that an object and its roles are the same."

Roles alone are beneficial to separate inherent and accidental characteristics of individuals. Encapsulating several related roles into a *role model* is another important aspect focused on in this paper. Results from the object-oriented software community [5,14] show that role models are an interesting unit of abstraction for mainly two reasons: First, role models focus on one specific concern of a domain, and hence, help in separating concerns. Second, role models are often reusable across domain boundaries because they can describe relations between individuals on a more general level than natural types can. We will focus on the second point, reuse of role models, in this work and show how role models serve as an ontological reuse unit.

An often discussed question is the relation of natural types and role types. Intuitively, natural types are related to role types via the "can play role" relationship, but the question is what semantics to associate. We will use the $N \rhd R$ notation for the "can play role" relation in this paper, where N is a natural type and R a role type. Sowa originally proposed to use the subsumption relationship ("IS-A") to explain $\rhd$, such that:

$$N \rhd R \;\equiv\; R \sqsubseteq N$$

where $\sqsubseteq$ represents the IS-A relationship. This interpretation is quite intuitive and works remarkably well, but not in all situations as we discuss in Section 4.3.

An alternative way of representing roles are separate individuals that are attached to individuals of natural types in some way. However, that contradicts role feature S14, since a role-playing individual would be split into at least two separate individuals. For a detailed discussion, the reader is referred to [17].

2.2 Description Logics and OWL

Description Logics (DLs) are a family of knowledge representation formalisms, where most members are sub-languages of first-order logics. DLs are used to capture the important *concepts* and relations (*roles* in DL parlance) between individuals of the modeled domain. We will refer to (binary) relations in DL as *dl-roles* to distinguish them from the conceptual role types. Concepts and dl-roles can be described by complex *concept* (resp. *dl-role*) *descriptions* using the construction operators available in the particular DL.

The most widely used DL is the one underlying the ontology language OWL DL. To simplify the presentation, we do not cover datatypes in this paper. An OWL DL *interpretation* is a tuple $I = (\Delta^I, \cdot^I)$ where the individual domain Δ^I is a nonempty set of individuals, and $\cdot^I$ is an individual interpretation function that maps (i) each individual name o to an element $o^I \in \Delta^I$, (ii) each concept name A to a subset $A^I \subseteq \Delta^I$, and (iii) each dl-role name RN to a binary relation $RN^I \subseteq \Delta^I \times \Delta^I$.

Valid OWL DL concept descriptions are defined by the DL syntax:

$$C ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \{o\} \mid \exists R.C \mid \forall R.C \mid \geqslant mR \mid \leqslant mR$$

The interpretation function $\cdot^{\mathcal{I}}$ is extended to interpret $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$. The concept $\top$ ($\bot$) is called *Thing* (*Nothing*) in OWL. The interpretation function can further be extended to give semantics to the remaining concept and dl-role descriptions (see [12] for details).

An OWL DL ontology consists of a set of *axioms*, including concept axioms, dl-role axioms and individual axioms.[2] A DL knowledge base consists of a TBox, an RBox and an ABox. A *TBox* is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$, where C, D are concept descriptions. An interpretation $\mathcal{I}$ satisfies $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An *RBox* is a finite set of dl-role axioms, such as dl-role inclusion axioms ($R \sqsubseteq S$). The kinds of dl-role axioms that can appear in an RBox depend on the expressiveness of the ontology language. An interpretation $\mathcal{I}$ satisfies $R \sqsubseteq S$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. An *ABox* is a finite set of individual axioms of the form $a : C$, called *concept assertions*, or $\langle a, b\rangle : R$, called *dl-role assertions*. An interpretation $\mathcal{I}$ satisfies $a : C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it satisfies $\langle a, b\rangle : R$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$.

Let C, D be concept descriptions, C is *satisfiable* wrt. a TBox $\mathcal{T}$ iff there exist an interpretation $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$; C subsumes D wrt. $\mathcal{T}$ iff for every interpretation $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A knowledge base Σ is *consistent* (*inconsistent*) iff there exists (does not exist) an interpretation $\mathcal{I}$ that satisfies all axioms in Σ.

Many people also use the Manchester OWL syntax [7], which is more user friendly for non-logicians.

3 A Role-Based Design Process for Ontological Reuse

The purpose of this section is to give an intuitive understanding of why role models are reusable entities and how they can be beneficial to ontology engineers. We use a running example to discuss consequences of a role modeling approach to ontology design. Furthermore, we discuss what the benefits of ontological role modeling are from a conceptual modeling point of view, and what the methodology of defining role-based ontologies is. Then, in Section 4, we discuss the semantics of such ontologies.

The examples are written in Manchester Syntax [7], which is not further described here but should be no problem to understand. We extend the syntax for the purpose of defining and composing role models; the keywords of the extended constructs are underlined (and in a different color).

Listing 1.1 shows an ontology that models a faculty, introducing main concepts such as *Professor*, *FacultyMember*, and *PhDStudent*. The faculty is managed by a board which is described in the role model in Listing 1.2. A board consists of board members that elect a chairman.[3] The chairman can appoint one of the members as secretary. The ontology in Listing 1.1 imports the board role model and can so use the concepts it defines. Concepts and properties defined in the role model are marked with $'$ to distinguish them from the concepts introduced in the base ontology.

Readers might ask why the board is described in a role model. The reason is that boards have a recognizable structure with a typical set of relationships that hold

[2] Individual axioms are called *facts* in OWL.

[3] A 'chairman' is here a person designated to preside over a meeting.

```
Ontology: http://ex.org/Faculty
  ImportRoles: http://ex.org/Board
  Class: FacultyMember
    CanPlay: BoardMember'
  Class: Professor
    SubclassOf: FacultyMember
    CanPlay: ChairMan'
  Class: PhDStudent
    SubclassOf: FacultyMember
  Individual: smith
    Types: Professor, Chairman'
  Individual: mike
    Types: PhDStudent, BoardMember'
```

Listing 1.1. Role-based ontology

```
RoleModel: http://ex.org/Board
  Role: BoardMember'
  Role: Chairman'
    SubclassOf: BoardMember' and
      electedBy' some BoardMember'
  Role: Secretary'
    SubclassOf: BoardMember'
  ObjectProperty: electedBy'
    Domain: Chairman'
    Range: BoardMember'
  ObjectProperty: appointedBy'
    Domain: Secretary'
    Range: Chairman'
```

Listing 1.2. Role model

between entities in that context, regardless of the particular underlying domain. It makes therefore sense to detach the description of the board from the faculty ontology.

The ontology in Listing 1.1 is made up of standard DL constructs, save the *ImportRoles* and *CanPlay* constructs. The meaning of the *ImportRoles* construct is the obvious, making the role model available to the ontology. The *CanPlay* constructs are crucial since they define the relations between the base ontology and the role model. We refer to such connecting statements as *bridge axioms*. The role model in Listing 1.2 makes use of two additional constructs, *RoleModel* and *Role* that have to obvious meaning. The URL of a *RoleModel* can be used to import it using the *ImportRoles* construct.

3.1 Methodology

Role modeling provides a methodology for developing ontologies, a methodology that we claim encourages good design and supports reuse. The following are the intuitive development steps that we propose for constructing a role-based ontology:

1. *Define base concepts.* Define a base ontology that contains the main concepts of the modeled domain. These concepts correspond to natural types of the domain. That is, each concept in the base ontology should be semantically rigid and non-founded. In our example, an ontology modeler would start by defining basic concepts of a faculty, such as *Professor* and *PhDStudent*. Notice that, in a different universe of discourse, *Professor* may itself be a role type (for example, for an underlying natural type *Person*).
2. *Identify role models.* Identify accidental or temporal relationships that individuals, abstracted by the base concepts, may participate in. Then, identify the contexts that those relationships appear in and what concepts (role types) are involved. Such contexts should be described in separate role models to be integrated into the base ontology.
   (a) If a role model for the desired relationships already exists, it can be reused.

(b) If no fitting role model exists, then define it. This involves capturing the important role types in the context and defining the relationships between them. There will exist relationships, since role types are by definition semantically non-rigid and founded. For instance, the board role model contains the role types *Chairman'* and *BoardMember'*, which are related by the *electedBy'* property.

To ensure reusability of role models they have to be self-contained. In particular, each property defined in a role model must have its domain and range restricted to a role type from the same role model. This guarantees that each individual that participates in such a property actually belongs to a role type of the role model.

3. *Define bridge axioms.* Describe how the identified role models should be integrated into the base ontology by defining appropriate bridge axioms. A bridge axiom can bind a role type to a natural type, assert that an individual belongs to a certain role type, or assert two individuals to be connected via a property that is part of a role model. For example, we connect *Professor* and *Chairman'* through a *CanPlay* axiom; the individual *smith* is asserted to be a *Chairman'*.

3.2 Benefits of Separating Role Models and Base Ontologies

We argue that it is beneficial to separate role models from base ontologies during the ontology design process. In particular, following the above methodology brings the following advantages:

– Modularization during development:
  - Base ontology development can focus on the main domain concepts and their hierarchical relations.
  - Role models can be defined, and refined, without necessarily focusing on the domain concepts, because role models typically transcend domains.
  - A role model focuses on a single context and important relationships holding between entities in this context.

– Reuse of role models:
  - As role models concentrate on a single concern, reuse is more likely than with complete ontologies that intermingle different concerns.
  - Role models constitute ontological modules. A base ontology can use role type names and property names of a role model, but not redefine them. Hence, a role model provides an interface, via the names of its role types and properties.

3.3 Reusing Role Models

The role-based ontology in Listing 1.3 demonstrates the reusability of the board role model from Listing 1.2. The role model is being deployed in a different setting, this time in an ontology modeling a company, instead of a university. Because the *concern* which is captured in the role model appears in both domains, it can be reused.

```
Ontology: http://ex.org/Company
  ImportRoles: http://ex.org/Board
  Class: President
    CanPlay: ChairMan'
  Class: VicePresident
    CanPlay: Secretary'
  Class: CompanyAdvisor
    CanPlay: BoardMember'
  Individual: donald
    Types: President, Chairman'
  Individual: jane
    Types: VicePresident, BoardMember'
```

Listing 1.3. Role-based ontology reusing the role model from Listing 1.2

4 Formalization and Semantics of Ontological Role Modeling

In Section 4.1 we formalize ontological roles and role models. Then, in Sections 4.2 to 4.3, we propose two possible semantics for the role modeling constructs used in the preceding section. The two semantics cover different aspects of role modeling and are realized by mapping role-based ontologies to different DL constructs.

4.1 Formalization of Role-Based Ontologies

We formalize role-based ontologies in three parts, based on the methodology described in Section 3. A base ontology only contains natural types. Role models define role types and their relationships. We use bridge axioms to combine a base ontology with role models into a role-based ontology.

Definition 1 (Base Ontology). *A* base ontology *is a finite set of axioms in some DL, capturing concepts that are assumed to correspond to natural types.*

A base ontology is assumed to capture concepts that provide semantic rigidity for individuals of the modeled domain. Naturally, any properties that inherently relate such concepts are also introduced, as are concrete individuals. We do not commit to a particular DL, since this definition is general enough to cover many DLs.

Definition 2 (Ontological Role Model). *An* ontological role model *is a TBox where each concept name is considered a role type. Each concept name must be "related" to another concept name, either via a dl-role, or by at least one axiom (e.g. a subsumption axiom). All dl-roles must be domain and range restricted to a type from the role model.*

The restriction of "related" concept names prevents role models from being divided into subparts with pairwise disjoint signatures.[4] If such a division is possible, the role model should be split into separate role models. Intuitively, this restriction ensures that a role model only describes one concern.

[4] As pointed out by a reviewer, what "related" means is not formalized. We recognize this formalization as important future work, but here stay with the intuitive notion, as described.

Definition 3 (Role-Based ontology). *A* role-based ontology *O is a triple $O = (\mathcal{N}, \mathcal{R}, \mathcal{B})$ where $\mathcal{N}$ is a* base ontology*, $\mathcal{R}$ is a finite set of* role models *and $\mathcal{B}$ a finite set of* bridge axioms*. The base ontology and the role models have pairwise disjoint signatures. The bridge axioms in $\mathcal{B}$ are of the form:*

1. *$N \triangleright R$ (*terminological bridge axiom*), or*
2. *$R(a)$ or $S(a,b)$ (*assertional bridge axiom*)*

where N is an arbitrary concept description, and a,b are individuals, in $\mathcal{N}$. R is a concept name (role type), and S a dl-role, in one of the role models in $\mathcal{R}$.

The $\triangleright$ symbol reads "can play" and specifies that instances of a natural type can play a role of a certain role type.

To be able to reuse existing tools, most importantly, reasoners, we define the semantics of role-based ontologies via reduction to the underlying DL. Thus, the reduction algorithm unambiguously defines the semantics of role-based ontologies by referring to the already understood model-theoretic semantics of DLs.

4.2 Conjunctive Role Modeling Semantics

The goal is to define a semantics for role-based ontologies that cover desirable properties of role modeling. As a minimal requirement, the semantics should cover the Steimann criteria S1, S2, S3, and S14 described in Section 2. More importantly, we need to account for the differentiation between natural and role types according to the distinction made by Guarino [3]. That is, natural types are semantically rigid, while role types are not, and do not provide identity for its instances. We will address this by acknowledging that individuals cannot only be instances of role types. This suggests that role types that are not explicitly related to some natural type should—by definition—be unsatisfiable (empty in all models of the ontology). Conveniently, unbound role types can easily be detected with standard ontology reasoners. Relations between natural and role types should explicitly be modeled by ontology engineers using terminological bridge axioms (that is, using the *CanPlay* construct).

The semantics of a role-based ontology $O = (\mathcal{N}, \mathcal{R}, \mathcal{B})$ is here given by a transformation to an ontology O' in the DL of $\mathcal{N}$ according to the following transformation:

$$\begin{aligned} O' = {} & \mathcal{N} \cup \mathcal{R} \\ & \cup \{R \sqsubseteq N | N \triangleright R \in \mathcal{B}\} \\ & \cup \mathcal{B} \setminus \{N \triangleright R | N \triangleright R \in \mathcal{B}\} \\ & \cup \{R \sqsubseteq \bot | R \in \mathcal{R} \wedge \neg \exists N : N \triangleright R \in \mathcal{B}\} \end{aligned}$$

As can be seen, the translation scheme consists of four steps:

1. *Integration.* The base ontology and the role models (with pairwise disjoint signatures) are combined.
2. *Terminological bridge axioms.* Here we use the semantics proposed by Sowa for realizing the $\triangleright$ relationship [15]. That is, if instances of a natural type N can play a role of a role type R, we specify R to be subsumed by N.

```
Ontology: http://ex.org/Faculty
  Class: FacultyMember
  Class: Professor
    SubclassOf: FacultyMember
  Class: PhDStudent
    SubclassOf: FacultyMember
  Class: BoardMember′
    SubclassOf: FacultyMember
  Class: Chairman′
    SubclassOf: BoardMember′ and
      electedBy′ some BoardMember′
      and Professor
```

```
Class: Secretary′
  SubclassOf: BoardMember′ and
              Nothing
ObjectProperty: electedBy′
  Domain: Chairman′
  Range: BoardMember′
ObjectProperty: appointedBy′
  Domain: Secretary′
  Range: Chairman′
Individual: smith
  Types: Professor, Chairman′
Individual: mike
  Types: PhDStudent, BoardMember′
```

Listing 1.4. Translation of a role-based ontology into the underlying ontology language

3. *Assertional bridge axioms.* All other bridge axioms are incorporated into O'. That is, all the assertional bridge axioms.
4. *Unbound role types*. Role types that are not related to any natural type through $\triangleright$ subsume $\perp$ (*Nothing* in OWL), that is, they are unsatisfiable.

We will illustrate the transformation using the example from Listings 1.1 and 1.2. Applying the transformation yields the ontology in Listing 1.4. This ontology only uses standard ontology constructs. It must be highlighted that the ontology in Listing 1.4 only captures the *meaning* of the ontology units from Listings 1.1 and 1.2. The ontology engineer is never expected to continue working on the "compiled" ontology. The abstractions gained through explicit role modeling are lost in the compilation step, making the resulting ontology more difficult to maintain. At the same time, because the compilation result is expressed in a standard DL, existing ontology reasoners can handle it directly.

For instance, the role type *Secretary'* is not bound to any natural type, and hence, defined as a subtype of *Nothing* ($\perp$). To understand the consequences of this translation, let us consider two scenarios:

1. Assume an additional assertion, stating that *mike* is an instance of *Secretary'*. As *Secretary'* is unbound, and thus, unsatisfiable, the resulting ontology would be inconsistent.
2. Assume another role type relies on instances of *Secretary'*. For instance, *Chairman'* could be a subclass of the concept *(assistedBy' some Secretary')*. As there can be no instances of *Secretary'*, *Chairman'* would also become unsatisfiable.

Obviously, our translation scheme for unbound role types helps in finding situations where role types are misused as natural types. The logical solution to repair our ontology for the two scenarios would be an additional bridge axiom *FacultyMember* $\triangleright$ *Secretary'*. In this case, *Secretary'* is no longer unsatisfiable and, as a consequence, the above illustrated inconsistencies do not occur.

Finally, let us come back to the four properties of Steimann from Section 2.1. As we define roles as concepts that can be described with all constructs of the underlying

DL, we fulfill S1. The second property, S2, is also supported, since we require that the context of each role type is captured in its surrounding role model. The second kind of bridge axiom in Definition 3 can be used arbitrarily often, hence, allowing one individual to be an instance of multiple role types (S3). The last property (S14) is supported as we do not represent roles as additional individuals but combine them with natural types using subsumption.

We refer to the above-described semantics as *conjunctive role modeling semantics* because a role type R played by different natural types $N_1, \ldots, N_n$ is interpreted as a subtype of the conjunction of the natural types, that is, $R \sqsubseteq N_1 \sqcap \ldots \sqcap N_n$. This follows immediately from defining $\triangleright$ using standard subsumption. While simple, this semantics does not come without problems from a role modeling perspective. We investigate this further in the next section.

4.3 Disjunctive Role Modeling Semantics

Another important role modeling feature from Steimann's overview paper [17] is the following:

S7 "*Objects of unrelated types can play the same role.* Although a fundamental observation [...] it is not acknowledged by all authors."

Although "not acknowledged by all authors", it is a rather intuitive and useful modeling notion. Imagine, for example, that we replace the class *Professor*, from Listing 1.1, with the two classes *FullProfessor* and *AssistantProfessor*. Imagine, furthermore, that they are declared to be disjoint (natural, since you cannot be both). Suppose we want to express that both kinds of professors can be chairmen in a board. Not only is this a natural thing to express, but doing so would also enable us to discuss the properties of a chairman (defined by *Chairman'*), regardless of which kind of professor it is. To do this, we would issue the following bridge axioms:

$$FullProfessor \triangleright Chairman'$$
$$AssistantProfessor \triangleright Chairman'$$

While the above axioms are intuitive to understand and write, the conjunctive role modeling semantics, based on Sowa's original interpretation of $\triangleright$, does not work as we perhaps would like. The reason is that the above gets interpreted as $Chairman' \sqsubseteq FullProfessor \sqcap AssistantProfessor$, which renders $Chairman'$ unsatisfiable. This is the case since the intersection of $FullProfessor$ and $AssistantProfessor$ is necessarily empty, since they are disjoint. In general, the conjunctive role modeling semantics can result in unexpected results when types that are not related via subsumption (e.g $FullProfessor$ and $AssistantProfessor$ in the above example) are related to the same role type via $\triangleright$.

To be able to address S7, we provide an alternative semantics by letting role types be subsumed by the *union* of all natural types their are bound to. For the above example, this results in $Chairman' \sqsubseteq FullProfessor \sqcup AssistantProfessor$, which in this case does not make $Chairman'$ unsatisfiable. We call this the *disjunctive role modeling semantics* and its formal realization is as follows:

```
...
Class: FullProfessor
  SubclassOf: FacultyMember
Class: AssistantProfessor
  SubclassOf: FacultyMember
...
Class: Chairman'
  SubclassOf: BoardMember' and
     electedBy' some BoardMember' and
     (FullProfessor or AssistantProfessor)
...
```

Listing 1.5. Translation of a role-based ontology into the underlying ontology language using disjunctive role modeling semantics

$$\begin{aligned}\mathcal{O}' = {} & \mathcal{N} \cup \mathcal{R} \\ & \cup \{R \sqsubseteq N_1 \sqcup \ldots \sqcup N_n | \{N_1, \ldots, N_n\} = \{N | N \rhd R \in \mathcal{B}\}\} \\ & \cup \mathcal{B} \setminus \{N \rhd R | N \rhd R \in \mathcal{B}\} \\ & \cup \{R \sqsubseteq \bot | R \in \mathcal{R} \wedge \neg\exists N : N \rhd R \in \mathcal{B}\}\end{aligned}$$

The above equations only differ from the corresponding equations for conjunctive semantic in the translation of terminological bridge axioms. Therefore, disjunctive semantics only differs from the conjunctive in cases where several natural types are bound to the same role type. In the case from the previous section with a single concept *Professor* bound to role type *Chairman'*, both semantics are equivalent. In contrast, the two semantics give different results if both *FullProfessor* and *AssistantProfessor* are bound to the role type *Chairman'*. While we obtain an inconsistency with the conjunctive semantics, disjunctive semantics allow for a consistent interpretation. The result is shown in Listing 1.5 (parts that are equal to Listing 1.4 are left out).

The disjunctive role modeling semantics satisfies S7, as well as the previously discussed role modeling requirements. Being able to connect unrelated, possibly disjoint, natural types to the same role type can be valuable from a modeling perspective. However, fulfilling S7 turns out to have a drawback: In contrast to standard DLs, the disjunctive semantics is non-monotonic. A logic is monotonic if adding a new axiom never falsifies assertions that were true before adding the axiom.

Lemma 1. *Ontological role modeling under disjunctive semantics is non-monotonic.*

The reason for this is that adding assertional bridge axioms can redefine previous knowledge. Consider the following role-based ontology $\mathcal{O} = (\mathcal{N}, \mathcal{R}, \mathcal{B})$, with:

$$\begin{aligned}\mathcal{N} &= \{FullProfessor \sqcap AssistantProfessor \sqsubseteq \bot, \\ &\quad FullProfessor(smith), AssistantProfessor(jones)\} \\ \mathcal{R} &= \{Chairman' = electedBy' \textbf{ some } BoardMember'\} \\ \mathcal{B} &= \{FullProfessor \rhd Chairman'\}\end{aligned}$$

Based on the disjunctive role modeling semantics we have $O \models \neg Chairman'(jones)$. But adding the bridge axiom $AssistantProfessor \rhd Chairman'$ to $\mathcal{B}$, this is not the case anymore, that is, $O \not\models \neg Chairman'(jones)$. As we have to retract knowledge when adding a bridge axiom, role modeling under the disjunctive semantics is non-monotonic.

5 Related Work

Reuse is an important part in the ontology development process. A structured reuse opportunity is presented by *upper-level ontologies* (e.g. [[10]]). Upper-level ontologies typically describe generic concepts related to notions such as time, space, matter, and have a high reuse value since they span many different domains. Role models also span different domains, but their reuse usage is different from upper-level ontologies. In general, upper-level ontologies reside "above" the base ontology and reuse is accomplished by declaring a base concept B as a subclass of the upper-level concept U, that is, $B \sqsubseteq U$ in DL syntax. Role models on the other hand reside "below" the base ontology and the reuse relationship is inverted from the upper-level one. That is, reuse is in general accomplished using axioms such as $R \sqsubseteq B$, where R is a role type from a role model. As such, role models are *complementary* to upper-level ontologies. The main incentive for upper-level ontologies is to achieve base ontology integration and reuse is a condition for its success. For role models, reuse *is* the incentive.

The OntoClean methodology proposed by Guarino [4] shares a number of ideas with our approach. The paper describes common misuses of the subsumption concept, for instance, to represent part/whole relations, instantiations, or meta-level relationships, and proposes to identify them using meta-properties for ontological classes. The two basic meta-properties of a class are *essence* and *rigidity*. Properties of a class belong to its essence, if they *must* hold for an instance (in contrast, for example, to properties of a role type, which *can* hold). Rigidity means that the properties of the class must hold for all instances. Our approach relates to this work as essential and rigid classes correspond to natural types, while non-essential and non-rigid classes are role types. Based on the meta-properties, Guarino proposes to impose constraints on subsumption relationships, for instance, forbidding a rigid concept to be subsumed by a non-rigid concept. This constraint is enforced in our approach by translating the terminological bridging axiom into $R \sqsubseteq C$, where a non-rigid concept R is always subsumed by a rigid concept C. Furthermore, OntoClean claims that each ontology has a backbone taxonomy with its rigid classes and their subsumption relationships. Such a backbone taxonomy roughly corresponds to our base ontology that exclusively consists of natural types.

The work in [19] proposes, similarly to us, to discriminate *role concepts* from *basic concepts* to overcome the gap between the recognition of different types of concepts and what is provided in standard ontology languages. The approach builds upon three notions: *role concepts*, equivalent to our role types, *potential players*, which roughly correspond to classes bound to a role type via a *CanPlay* relationship, and *role-holder*, that is, instances actually playing a role. The authors argue for two distinct type hierarchies and emphasize the relation of a role to its context. Furthermore, the paper describes compound roles that are built from primitive roles, realizing ideas that are similar to roles playing roles as in [17]. In contrast to our approach, the authors implement roles in their own ontology framework, including an ontology language and custom-built tools.[5] Instead, we propose to embed roles into existing languages using syntactic extensions that can be translated into the underlying ontology language.

6 Conclusions and Outlook

Ontologies are increasingly often applied in real-life scenarios, where they are used for modeling large knowledge domains. Perhaps the most prominent example to name here is the Gene Ontology [20] with its almost 25.000 terms (as of January 2008). To successfully develop and maintain such large ontologies, powerful modularization and reuse techniques are required.

In this paper we have discussed the notion of roles as explicit modeling constructs, that has been discussed in the domain of conceptual modeling for some time now, but never really utilized in ontology languages. We have shown how making roles explicit concepts—instead of encoding them implicitly in dl-roles—enables us to encapsulate role models and reuse them in different ontologies, even across domain boundaries. A role model in this context is an ontology consisting of role concepts and relationships between them. We have shown how to construct role-based ontologies from a base ontology, a role model and a set of bridging axioms relating natural concepts from the base ontology to role concepts from the role model via a *CanPlay* relationship. The semantics of this new relationship has been defined by translating role-based ontologies to equivalent ontologies in a standard DL. In this paper, we have discussed two such semantics: *conjunctive semantics* and *disjunctive semantics*. The former is simpler, but does not allow individuals of disjoint natural concepts to play the same role, which may be counter intuitive to ontology designers. The latter semantics allows such situations, but at the cost of a non-monotonic logic. It is at this stage not possible to recommend one of the semantics as canonical—they serve different purposes. We first need to study how role models are used and applied in practice to better understand which semantics is more intuitive to ontology engineers.

In conclusion, we suggest to integrate explicit role concepts with ontology languages, such as OWL, as they offer a unique reuse and modularization opportunity that goes beyond currently available mechanisms for ontology reuse. Role models form *components*; their role types define a component interface enabling to check correct usage of the component—for example, every used role type must be assigned to some base concept. A notion of components that allows checking for correct usage is lacking in today's ontology languages. This gap can be closed by role modeling. Role modeling goes beyond upper-level ontology reuse, as it allows more specific ontologies to be reused and also allows reusing multiple role models within one ontology. Role models are, therefore, an important tool in every ontology designer's tool box.

Of course, more work is needed. We would like to perform a larger case study of applying role models to refactoring, modularizing, and partially reusing a large ontology. Role modeling should be supported by ontology modeling tools that can transform the role-based ontologies before applying reasoners. Furthermore, it will be interesting to

[5] `http://www.hozo.jp`

study other applications of role models in ontologies—for example, where ontologies are used for describing situations in an action calculus [9].

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Cuenca-Grau, B., Motik, B.: OWL 2 Web Ontology Language: Model-theoretic semantics. W3C Working Draft (2008), http://www.w3.org/TR/owl2-semantics/
3. Guarino, N.: Concepts, attributes and arbitrary relations - Some linguistic and ontological criteria for structuring knowledge bases. Dat. Knowl. Eng. 8(3), 249–261 (1992)
4. Guarino, N., Welty, C.A.: Evaluating ontological decisions with OntoClean. Communications of the ACM 45(2), 61–65 (2002)
5. Herrmann, S.: Object Teams: Improving modularity for crosscutting collaborations. In: Net Object Days (2002)
6. Herrmann, S.: A precise model for contextual roles: The programming language ObjectTeams/Java. Applied Ontology 2(2), 181–207 (2007)
7. Horridge, M., Patel-Schneider, P.F.: Manchester syntax for OWL 1.1. In: International Workshop OWL: Experiences and Directions (OWLED 2008) (2008)
8. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
9. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Reasoning about actions using description logics with general TBoxes. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 266–279. Springer, Heidelberg (2006)
10. Niles, I., Pease, A.: Towards a standard upper ontology. In: International conference on Formal Ontology in Information Systems (FOIS 2001), pp. 2–9. ACM Press, New York (2001)
11. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation (2004), http://www.w3.org/TR/owl-semantics/
12. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, W3C, W3C Recommendation (February 2004)
13. Pradel, M., Henriksson, J., Aßmann, U.: A good role model for ontologies: Collaborations. In: International Workshop on Semantic-Based Software Development (2007)
14. Reenskaug, T., Wold, P., Lehne, O.: Working with Objects, The OOram Software Engineering Method. Manning Publications Co. (1996)
15. Sowa, J.: Using a lexicon of canonical graphs in a semantic interpreter, pp. 113–137. Cambridge University Press, Cambridge (1988)
16. Sowa, J.F.: Conceptual structures: information processing in mind and machine. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (1984)
17. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data Knowledge Engineering 35(1), 83–106 (2000)
18. Steimann, F.: The role data model revisited. Roles, an interdisciplinary perspective. In: AAAI Fall Symposium (2005)
19. Sunagawa, E., Kozaki, K., Kitamura, Y., Mizoguchi, R.: Role organization model in Hozo. In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS (LNAI), vol. 4248, pp. 67–81. Springer, Heidelberg (2006)
20. The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. Nature Genetics, 25(1), 25–29 (May 2000)

Enhancing a DLP System for Advanced Database Applications

G. Terracina, E. De Francesco, C. Panetta, and N. Leone

Dipartimento di Matematica, Università della Calabria,
I-87036 Rende (CS), Italy
{terracina,defrancesco,panetta,leone}@mat.unical.it

Abstract. Disjunctive logic programming under answer set semantics (DLP, ASP) is a powerful rule-based formalism for knowledge representation and reasoning. The language of DLP is very expressive, and allows to model also advanced knowledge-based tasks arising in modern application-areas like, e.g., information integration and knowledge management. The recent development of efficient systems supporting disjunctive logic programming, has encouraged the usage of DLP in real-world applications. However, despite the high expressiveness of their languages, the success of DLP systems is still dimmed when the applications of interest become data intensive (current DLP systems work only in main memory) or they need the execution of some inherently procedural sub-tasks. The main goal of this paper is precisely to improve efficiency and usability of DLP systems in these contexts, for a full exploitation of DLP in real-world applications.

We develop a DLP system which *(i)* carries out as much as possible of the reasoning tasks in mass memory without degrading performances, allowing to deal with data-intensive applications; *(ii)* extends the expressiveness of DLP language with external function calls, yet improving efficiency (at least for procedural sub-tasks) and knowledge-modeling power; *(iii)* incorporates an optimization strategy, based on an unfolding technique, for efficient query answering; *(iv)* supports primitives allowing to integrate data from different databases in a simple way.

We test the system on a real data-integration application, comparing its performance against the main DLP systems. Test results are very encouraging: the proposed system can handle significantly larger amounts of data than competitor systems, and it is also faster in response time.

1 Introduction

Current capabilities of generating and collecting data are increasing rapidly. The wide-spread use of internet applications for most commercial activities, the computerization of many business and government transactions, and the advances in data collection tools have provided us with huge amounts of data. This explosive growth in data and databases has generated an urgent need for new techniques and tools that can intelligently and automatically infer useful information and knowledge from available data.

Knowledge representation and reasoning capabilities required in these contexts could be provided also by a powerful rule-based formalism like disjunctive logic programming under answer set semantics (DLP, ASP).

The recent development of efficient DLP systems, like DLV [14], Cmodels [10], Gnt2 [12], and ClaspD [8], has renewed the interest for DLP in modern application areas. As an example, DLV has been recently exploited for data-integration [7], ontology specification [18], and ontology querying [11].

However, current DLP systems present two main drawbacks in real world scenarios: they are not capable of handling data intensive applications (they work in main memory only) and they are not well suited for modelling inherently procedural problems.

Recently, we presented a database-oriented variant of DLV, namely DLV^{DB} [20], representing a first step towards overcoming these drawbacks. In fact, [20] carries out all of its tasks in mass memory, thus enabling data intensive applications, but only for limited forms of reasoning (only disjunction free, stratified programs are allowed).

The goal of this work is to enhance DLV^{DB} features to improve its efficiency and usability in the contexts outlined above, for an effective exploitation of DLP in real world scenarios. The proposed enhancements include:

– Full support to disjunctive datalog with unstratified negation, and aggregate functions;
– Extension of DLP with external function calls, particularly suited for solving inherently procedural sub-tasks but also for improving knowledge-modelling power;
– An evaluation strategy devoted to carry out as much as possible of the reasoning tasks in mass memory, thus enabling complex reasonings in data intensive applications without degrading performances;
– An optimization strategy oriented to improve query answering;
– Primitives to integrate data from different databases.

In order to make the above enhancements possible, various challenges had to be faced:

1. Data intensive applications usually must access, and modify, data stored in enterprise databases and these should be accessed also by other applications.
2. Evaluating the stable models of a program directly in mass-memory data-structures, could be highly inefficient.
3. Using the main memory to accommodate both the input data (hereafter, EDB) and the inferred data (hereafter, IDB) is usually impossible for data intensive applications due to the limited amount of available main memory.

Note that, from points 2. and 3. it comes out that some amount of data *must* be loaded in main memory, but this should be as small as possible.

In order to face challenge 1. DLV^{DB} is interfaced with external databases via ODBC. ODBC allows a very straightforward way to access and manipulate data over, possibly distributed, databases. Note that challenge 1. makes systems integrating proprietary DBMSs less effective.

As far as challenge 2. is concerned we adopt a mixed strategy, which is detailed in Section 2; intuitively, the evaluation can be divided in two distinct phases: the grounding and the model generation. Grounding is completely performed in the database, whereas the model generation is carried out in main memory; this allows also to address challenge 3. In fact, in several cases, only a small portion of the ground program is actually needed for the model generation phase, since most of the inferred data is "stable" and belongs to every stable model (and is already derived during the grounding phase).

We have dedicated special attention also to efficiency; in fact, while language extensions and mass memory evaluations usually tend to degrade systems efficiency, our implementation presents comparable, and in several cases even better, performances than competitor main memory systems, yet allowing the handling of the highest amounts of data. The proposed system has been in fact compared with state-of-the-art ASP systems on a real data-integration scenario. Test results, reported in the paper, show that DLV^{DB} is well suited for data intensive applications both for time and space requirements.

The plan of the paper is as follows. Section 2 describes the overall characteristics of the proposed system, and its database oriented evaluation strategy. Section 3 describes the application scenario adopted for our tests, whereas Section 4 presents benchmarks and results. Finally, Section 5 provides some final remarks.

2 System Description

The language supported by the proposed system is disjunctive datalog, extended with functions and aggregates. Syntax and semantics of this language are described in detail in [5,6]. In the following we focus on our implementation of the language, recalling just the syntax of rules, and the main syntactic conditions to be respected.

Rules accepted by the system have the form

$$\alpha_1 \vee \cdots \vee \alpha_k \text{ :- } \beta_1, \ldots, \beta_n, \text{not}\, \beta_{n+1}, \ldots, \text{not}\, \beta_m. \tag{1}$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$, are ordinary atoms, and $\beta_1, \ldots, \beta_m$ are (ordinary, external, or aggregate) atoms. External and aggregate atom predicate names are conventionally preceded by "#". Rules with $k = 0$ and $m > 0$ are called *constraints*, whereas rules such that $k = 1$ and $m = 0$ are called *facts*.

An example of external atom could be #concat(X,Y,Z), which takes two strings X and Y as input and returns a string Z corresponding to the concatenation of X and Y. Examples of aggregate functions are #`count` (number of terms) and #`sum` (sum of rational numbers).

Functions introduced in the program by external atoms are expected to be defined as scalar stored functions in the database coupled with DLV^{DB}; in fact, as it will be clear in the following, DLV^{DB} performs most of its evaluations directly on a working database specified by the user. Moreover, programs must be value-invention restricted (cfr. [5]), i.e. new values possibly introduced by external atoms must not propagate through recursion; this avoids the generation of infinite-sized answer sets.

Stored functions in databases can return only one scalar value; as a consequence, DLV^{DB} adopts the convention that the last variable of the external atom corresponds to the result returned by the function call, whereas all the other variables are the inputs for the stored function.

The evaluation strategy implemented in our approach puts its basis on the sharp distinction existing between the grounding of the input datalog program and the generation of its stable models. Then, two distinct approaches can be adopted whether the input program is non disjunctive and stratified (in this case everything can be evaluated on the DBMS) or not. Details are provided next.

Evaluation of non disjunctive stratified programs. It is well known that if a program is non disjunctive and stratified, it has a unique stable model corresponding exactly to its ground instantiation. The evaluation of these kinds of program has been already addressed in our previous version of DLV^{DB}. Intuitively, it consists in the translation of each datalog rule in a corresponding SQL statement and in the composition of a suitable query plan on the DBMS; the evaluation of recursive rules is carried out with an improved semi-naïve approach. A detailed description of the approach is out of the scope of this paper; the interested reader is referred to [20] for details.

This paper, however, extends the language previously supported by DLV^{DB} to allow also for external atoms. Note that since the grounding phase instantiates all the variables, there is no need to invoke again the functions associated with external atoms after the grounding (this is true even for disjunctive or non stratified programs). As a consequence, the handling of external atoms can be carried out completely during the grounding and, hence, within the SQL statements generated from the datalog rules.

Now recall that, by convention, given an external atom $\#f(X_1, \ldots, X_n, O)$ used in a rule r, only the last variable O can be considered as an output parameter, while all the other variables must be intended as input for f. This corresponds to the function call on the database $f(X_1, \ldots, X_n) = O$. Moreover, O can be: *(i)* bound to another variable in the body of r, *(ii)* bound to a constant, *(iii)* a variable of r's head. Then, in the SQL statement corresponding to r, a function call is introduced in the WHERE part to implement cases *(i)* and *(ii)* and in the SELECT part to implement case *(iii)*.

As an example, consider the rule:

```
mergedNames(ID, Name) :- person(ID, FirstName, LastName),
                                   #concat(FirstName, LastName, Name).
```

This rule belongs to case *(iii)* above and is translated into:

```
INSERT INTO mergedNames
(SELECT person.ID, concat(person.FirstName,person.LastName) FROM
person);
```

Cases *(i)* and *(ii)* are handled analogously.

Evaluation of disjunctive programs with unstratified negation. In presence of disjunctive rules or unstratified negation in a program $\mathcal{P}$, the ground instantiation of $\mathcal{P}$ is not sufficient to compute its stable models. Then, grounding and model generation phases must both be handled.

The evaluation strategy we adopt in this paper carries out the grounding completely in the database, by the execution of suitable SQL queries. This phase generates two kinds of data: ground atoms (facts) valid in every stable model (and thus not requiring further elaboration in the model generation phase) and ground rules, summarizing possible values for a predicate and the conditions under which these can be inferred.

Facts compose the so called *solved* part of the program, whereas ground rules form the *residual program*, not completely solved by the grounding. As previously pointed out, one of the main challenges in our work is to load the smallest amount of information as possible in main memory; consequently, the residual program generated by the system should be as small as possible.

Model generation is then carried out in main memory with the technique described in [14].

Definition 1. *Let p be a predicate of a program $\mathcal{P}$, p is said to be* unsolved *if:* (i) *it is in the head of a disjunctive rule;* (ii) *it is the head of at least one rule involved in unstratified negation;* (iii) *the body of a rule having p as head contains at least one unsolved predicate. p is said to be* solved *otherwise.*

In our evaluation strategy, a ground solved predicate is associated with facts only in the ground program and, thus, with *certainly-true* values, i.e. values true in every stable model. On the contrary, a ground unsolved predicate p may be defined by both facts (certainly-true values) and ground rules; the latter identify *possibly-true* values for p, i.e. the domain of values p may assume in stable models.

Given an unsolved predicate p we indicate the set of its certainly-true values as p^s and the set of its possibly-true values as p^u.

Example 1. Consider the simple program: q(1,2). p(3). p(X) $\vee$ p(Y) :- q(X,Y). Here q is a solved predicate, whereas p is an unsolved predicate; in particular, $q(1,2)$ is a certainly-true value for q, $p(3)$ is a certainly-true value for p, whereas $p(1)$ and $p(2)$ are possibly-true values of p. Then, $p^s = \{3\}$, whereas $p^u = \{1,2\}$.

As previously pointed out, rules having an unsolved predicate may generate ground rules in the instantiation. Since we are interested in generating the smallest residual program as possible, ground rules are “epurated” of certainly-true values.

Definition 2. *A* simplified ground rule *(g-rule in the following) of a program $\mathcal{P}$ is a ground rule not involving any certainly-true values of $\mathcal{P}$.*

Example 2. From the previous example, the (only) ground rule that can be generated is $p(1) \vee p(2)$:- $q(1,2)$. However this is not a simplified rule since it involves $q(1,2)$ which is a certainly-true value. Then, the corresponding g-rule is simply $p(1) \vee p(2)$.

It is now possible to illustrate the evaluation strategy implemented in our system. Consider a program $\mathcal{P}$ composed of non ground rules of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \text{ :- } \beta_1, \ldots, \beta_n, \text{not}\, \beta_{n+1}, \ldots, \text{not}\, \beta_m, \gamma_1, \ldots, \gamma_p, \text{not}\, \gamma_{p+1}, \ldots, \text{not}\, \gamma_q. \tag{2}$$

where β_i (resp. γ_j) are solved (resp. unsolved) predicates. The evaluation is carried out in five steps:

Step 1. Translate $\mathcal{P}$ in an equivalent program $\mathcal{P}'$;
Step 2. Translate each rule of $\mathcal{P}'$ in a corresponding SQL statement;
Step 3. Compose and execute the query plan of statements generated in Step 2 on the DBMS;
Step 4. Generate the residual program and load it in the Model Generator of DLV;
Step 5. Execute the residual program in main memory and show the results.

Step 1. The objective of this step is to "prepare" rules of $\mathcal{P}$ to be translated in SQL almost straightforwardly, in order to generate a residual programs as small as possible. In more detail, for each rule r in $\mathcal{P}$ three kinds of rule are generated:

A. If the head of r has one atom only ($k = 1$), a rule (hereafter denoted as A-rule) is created for deriving only certainly-true values of r's head; note that if $k > 1$ no certainly-true values can be derived from r.
B. A set of rules (hereafter, B-rules) supporting the generation of the g-rules of r. The heads of these rules contain both the variables of unsolved predicates in the body of r and the variables in the head of r. Ground values obtained for these variables with B-rules are then used to instantiate r with possibly-true values only.
C. A set of rules (hereafter, C-rules) for generating the set of possibly-true values of unsolved predicates as projections on B-rules obtained previously.

Given a generic rule defined as (2), the corresponding A-rule have the form:

$$\alpha_1^s \text{ :- } \beta_1, .., \beta_n, \text{not}\, \beta_{n+1}, .., \text{not}\, \beta_m, \gamma_1^s, .., \gamma_p^s, \text{not}\, \gamma_{p+1}^s, \text{not}\, \gamma_{p+1}^u, .., \text{not}\, \gamma_q^s, \text{not}\, \gamma_q^u. \tag{3}$$

where for positive unsolved predicates only certainly-true values ($\gamma_1^s, \ldots, \gamma_p^s$) are considered, whereas for negated unsolved predicates both certainly-true and possibly-true values ($\gamma_{p+1}^s, \ldots, \gamma_q^s$, $\gamma_{p+1}^u, \ldots, \gamma_q^u$) must be taken into account.

Example 3. Consider the following program, which will be exploited as a running example throughout the rest of the section:

r1: q(1,2). r2: p(Y,X) $\vee$ t(X) :- q(X,Y). r3: q(X,Y):- p(X,Y), not t(X).

Here both p, q, and t are unsolved. The A-rules derived for this program are:

r1.A: q^s(1,2).
r3.A: q^s(X,Y):- p^s(X,Y), not t^s(X), not t^u(X).

where rule $r2$ does not contribute since it is disjunctive and cannot generate certainly-true values.

B-rules play a key role in our approach; in fact, they allow the generation of the residual program. In particular, their role is to identify the set of values for variables in unsolved predicates of the body of r, generating possibly-true values of the head of r. Then, r is seen as a template for generating its g-rules, and ground values derived by the corresponding B-rules are used to instantiate r.

Note that in order to generate a possibly true value for a normal rule, at least one possibly true value must be involved in its body, whereas disjunctive rules always generate possibly-true values. Moreover, in order to properly generate g-rules (i.e. ground rules involving possibly-true values only) the system must be able to track, for each truth value of a B-rule, which predicates of r contributed with a certainly-true value and which ones with a possibly-true value.

In our approach, this issue is addressed by first labelling each unsolved predicate γ_j of r alternatively with a 0 or with a 1, where a 0 indicates to take its γ_j^s, whereas a 1 indicates to consider its γ_j^u. Then, each binary number between 1 and 2^q-1 for normal rules and between 0 and 2^q-1 for disjunctive rules[1] corresponds to a labelling stating the combination of values to be considered. For each labelling, a corresponding B-rule is generated starting from the definition of r and substituting each unsolved predicate γ_j with γ_j^s (resp., γ_j^u) if the corresponding label is 0 (resp., 1).

The only exception is caused by negated unsolved predicates. In fact, if γ_j is negated and labelled with a 1, it must be put in the B-rule without negation. In fact, negated certainly-true values surely invalidate the satisfiability of the g-rule, whereas negated possibly-true values may invalidate the rule only if the model generator sets them to true.

It is worth pointing out that our labelling approach, makes significantly easier the generation of simplified ground rules; in fact, it is sufficient to consider only the values of predicates labelled with 1 and not derived to be certainly-true by other rules.

Finally, in order to allow a proper reconstruction of g-rules from B-rules, a mapping between the variables of the B-rules and the variables of r is maintained.

Example 4. From rules $r2$ and $r3$ introduced in the previous example, the following B-rules are derived. Original rules are re-proposed in parenthesis to simplify the comprehension; labels are reported in rule names. Variable mapping is trivial and not reported.

	(r2: p(Y,X) $\vee$ t(X) :- q(X,Y).)
r2.B(0):	B-rule_r2(X,Y):- q^s(X,Y).
r2.B(1):	B-rule_r2(X,Y):- q^u(X,Y).
	(r3: q(X,Y):- p(X,Y), not t(X).)
r3.B(01):	B-rule_r3(X,Y):- p^s(X,Y), t^u(X).
r3.B(10):	B-rule_r3(X,Y):- p^u(X,Y), not t^s(X).
r3.B(11):	B-rule_r3(X,Y):- p^u(X,Y), t^u(X).

[1] Recall that q is the number of unsolved predicates in the body of r.

Finally, C-rules are simple projections on the B-rule heads over the attributes of the corresponding predicate.

Example 5. From rules $r2$ and $r3$ introduced previously and from the corresponding B-rules the system generates:

```
r2.C_p:    p^u(Y,X):- B-rule_r2(X,Y).
r2.C_t:    t^u(X):- B-rule_r2(X,Y).
r3.C_q:    q^u(X,Y):- B-rule_r3(X,Y).
```

Note that in the example above, the same B-rule predicate (B-rule_r2) is used to generate possibly-true values of two predicates (p^u and t^u); this follows directly from the fact that $r2$ is a disjunctive rule involving p and t.

Step 2. Translation of the rules obtained in Step 1. into SQL is carried out with the technique already presented in [20] for non disjunctive and stratified programs. As an example, rule $r3.A$ introduced above is translated into[2]:

```
INSERT INTO q^s (SELECT p^s.att_1, p^s.att_2, FROM p^s
WHERE p^s.att_1 NOT IN (SELECT * FROM t^s)
AND p^s.att_1 NOT IN (SELECT * FROM t^u))
```

Step 3. In order to compile the query plan, the dependency graph D associated with $\mathcal{P}$ is considered [15]. In particular, D allows the identification of a partially ordered set $\{\text{Comp}_i\}$ of program components where lower components must be evaluated first.

Then, given a component Comp and a rule r in Comp, if r is not recursive, then the corresponding portion of query plan is as follows[3]: *(1)* evaluate (if present) the A-rule associated with r; *(2)* evaluate each B-rule obtained from r; *(3)* for each predicate in the head of r evaluate the corresponding C-rule.

If r is recursive, the portion of query plan above must be included in a fix-point semi-naïve evaluation, as described in [20].

Step 4 and 5. The generation of the residual program requires the analysis of values derived by B-rules only. Then, for each rule r and each corresponding B-rule (say, r.B(L)), first predicates having label 0 in L are purged from r, then r is instantiated with values of r.B(L); during this phase a further check is carried out to verify if some predicate value has been derived as certainly-true by other rules. In this case the predicate is removed from the g-rule for that instance. The residual program is then loaded in main memory for the generation of stable models. Note that each answer set found on this residual program shall be enriched with certainly-true values determined during the grounding.

[2] Here and in the following we use the notation x.att$_i$ to indicate the i-th attribute of the table x. Actual attribute names are determined at runtime.

[3] Here, for simplicity of exposition, we refer to rules, indicating that the corresponding SQL statements must be evaluated on the database.

Example 6. The residual program generated for our running example is:

p(2,1) v t(1). p(1,2) v t(2) :- q(2,1). q(2,1):- p(2,1), not t(2).

Note that the first g-rule does not involve q since it derives from r2.B(0), having $q(1,2)$ as certainly-true value.

Query oriented optimizations. Query answering under the cautious (resp., brave) semantics basically consists in determining the set of truth values of predicates involved in the query, holding in all (resp., at least one) answer sets of the program. It is well known that answer set programming implements a bottom-up evaluation strategy which is not well suited for query answering purposes, especially when the query contains constants, where top-down approaches usualy perform better. To this purpose, many strategies have been developed so far to simulate the top-down evaluation within the answer sets setting (see, e.g., magic sets [2,16]).

DLV^{DB} incorporates query oriented optimizations based on both magic sets (already incorporated in the earlier version of the system) and query unfolding. The latter optimization basically consists in an unfolding of the query whose final desiderata is the rewriting of the query in terms of EDB predicates only. Clearly, when dealing with disjunctive and unstratified programs, this is not always possible and, under some conditions, not convenient; however, there are several situations in which unfolding can push down selections and sub-queries onto the EDB, thus allowing significant performance improvements.

The algorithm for the unfolding is quite complex and, due to space constraints, we can not show it here; the interested reader is referred to [3] for details.

Primitives for integrating data from different databases. As previously pointed out, DLV^{DB} can be coupled with a DBMS to carry out mass memory evaluations. Actually, the system provides more involved kinds of primitives to access data residing on different databases. These can be specified by the user with some auxiliary directives whose grammar is shown in Figure 1.

Intuitively, the user must specify the working database and can specify a set of table definitions; note that each specified table must be mapped into one of the

```
Auxiliary-Directives ::= Init-section [Table-definition]+ [Query-Section]? [Final-section]*
Init-Section ::=USEDB DatabaseName:UserName:Password [System-Like]?.
Table-definition ::=
   [USE TableName [( AttrName [, AttrName]* )]? [AS ( SQL-Statement )]?
   [FROM DatabaseName:UserName:Password]?
   [MAPTO PredName [( SqlType [, SqlType]* )]? ]?.
   |
   CREATE [VIEW] TableName [( AttrName [, AttrName]* )]?
   [MAPTO PredName [( SqlType [, SqlType]* )]? ]?
   [KEEP_AFTER_EXECUTION]?.]
Query-Section ::= QUERY TableName.
Final-section ::=
   [DBOUTPUT DatabaseName:UserName:Password.
   |
   OUTPUT [APPEND | OVERWRITE]? PredName [AS AliasName]?
   [IN DatabaseName:UserName:Password.]
System-Like ::= LIKE [POSTGRES | ORACLE | DB2 | SQLSERVER | MYSQL]
```

Fig. 1. Grammar of the auxiliary directives

program predicates. Facts can reside on separate databases or they can be obtained as views on different tables. The `USE` and `CREATE` options can be exploited to specify input and output data. Finally, the user can choose to copy the entire output of the evaluation or parts thereof in a database different from the working one.

Of particular interest is the `VIEW` option of the `CREATE` statement (not available in the earlier version of DLV^{DB}); this can be used to specify that tables associated with intermediate predicates (i.e. corresponding to neither input nor output data) should be maintained virtual. This possibility may provide both space saving and performance improvements, especially for those programs having a hierarchical structure.

Example 7. Assume that a travel agency asks to derive all the destinations reachable by an airline company either by using its aircrafts or by exploiting code-share agreements. Suppose that the direct flights of each company are stored in a relation `flight_rel(Id, FromX, ToY, Company)` of the database `dbAirports`, whereas the code-share agreements between companies are stored in a relation `codeshare_rel (Company1, Company2, FlightId)` of an external database `dbCommercial`; if a code-share agreement holds between the company $c1$ and the company $c2$ for $flightId$, it means that the flight $flightId$ is actually provided by an aircraft of $c1$ but can be considered also carried out by $c2$. The DLP program that can derive all the connections is:

```
(1) destinations(FromX, ToY, Comp) :- flight(Id, FromX, ToY, Comp).
(2) destinations(FromX, ToY, Comp) :- flight(Id, FromX, ToY, C2), codeshare(C2,Comp, Id).
(3) destinations(FromX, ToY, Comp) :- destinations(FromX, T2, Comp),
                                      destinations(T2, ToY, Comp).
```

In order to exploit data residing in the above mentioned databases, we should map the predicate *flight* to the relation `flight_rel` of `dbAirports` and the predicate *codeshare* to the relation `codeshare_rel` of `dbCommercial`. Finally, we have to map the predicate *destinations* to the relation `composedCompanyRoutes` of `dbTravelAgency`. To this purpose, the auxiliary directives shown in Figure 2 should be used.

```
USEDB dlvdb:myname:mypasswd.
USE flight_rel (Id, FromX, ToY, Company) FROM dbAirports:airportUser:airportPasswd
MAPTO flight (integer, varchar(255), varchar(255), varchar(255)).
USE codeshare_rel (Company1, Company2, FlightId) FROM dbCommercial:commUser:commPasswd
MAPTO codeshare (varchar(255), varchar(255), integer).
CREATE destinations_rel (FromX, ToY, Company)
MAPTO destinations (varchar(255), varchar(255), varchar(255)) KEEP_AFTER_EXECUTION.
```

Fig. 2. Auxiliary directives for Example 7

3 Application to Data Integration

Data integration systems provide a transparent access to different and possibly distributed sources. The user is provided with a uniform view of available information by the so-called *global schema*, which queries can be posed upon. The integration system is then in charge of accessing the single sources separately and merging data relevant for the query, guided by mapping rules that specify relationships holding between the sources and the global schema [1,13].

The global schema may contain integrity constraints (such as key dependencies, inclusion dependencies, etc.). The main issues in data integration arise when original sources independently satisfy the integrity constraints but, when they are merged through the mappings, they become inconsistent. As an example, think to the lists of students of two universities; each student has an unique ID in his university, but two different students in different universities may have assigned the same ID. Clearly, when they are loaded in a global database merging students lists, it is likely that the key constraint on student IDs of the global schema will be violated.

Most of the solutions to these problems are based on database repair approaches. Basically, a repair is a new database satisfying constraints of the global schema with minimal differences from the source data. Note that multiple repairs can be singled out for the same database. Then, answering queries over globally inconsistent sources consists in computing those answers that are true in every possible repair; these are called *consistent* answers in the literature.

DLP under ASP is a powerful tool in this context, as demonstrated for example by the approaches formalized in [1,4,7]. In fact, if mappings and constraints on the global schema are expressed as DLP programs, and the query Q as a union of conjunctions on the global schema, the database repairs correspond to the stable models of the program, and the consistent answers to Q correspond to the answers of Q under cautious reasoning.

As an example, the approach proposed in [7] consists in first retrieving as much information as possible from the sources, and then building the repairs by removing the minimal amount of inconsistent data and adding the minimal amount of missing data.

Example 8. To have an intuition on the repair approach proposed in [7] for handling key constraints, consider two sources s1(SID, StudentName) and s2(SID, StudentName), storing the students of two universities, and assume that the global schema is designed so as to merge these lists. The program defining the mappings for the global relation studentG and handling the key constraint over SID is:

studentD(SID,SName):- s1(SID,SName).
studentD(SID,SName):- s2(SID,SName).
studentG(SID,SName):-studentD(SID,SName), not $\overline{\textit{student}}$(SID,SName).
$\overline{\textit{student}}$(SID,SName1) v $\overline{\textit{student}}$(SID,SName2):- studentD(SID,SName1),
                                        studentD(SID,SName2), SName1≠SName2.

Here the first two rules load all possible data from the sources, whereas the third one avoids to put conflicting tuples in the global relation studentG. Note that the disjunctive rule allows the generation of the various repairs by singling out the conflicting tuples.

Now, assume that s1 contains {s1(1234, Jhon), s1(2345, Andrew)} and s2 contains {s2(1234, David)}. There is globally a conflict between Jhon and David because they have the same ID. Then, there are two repairs for studentG, namely {studentG(1234,Jhon), studentG(2345, Andrew)} and {studentG(1234,David), studentG(2345, Andrew)}. If the user poses the query Q1(SName):-studentG (SID,SName), the only consistent answer is: {Andrew}, but if the user asks for Q2(SID):- studentG(SID,SName), the consistent answers are: {1234,2345}.

4 Experiments and Benchmarks

In this section we describe the tests we carried out in querying inconsistent and incomplete data. We exploited a data integration framework developed in the INFOMIX project (IST-2001-33570) [7] which integrates real data from a university context.

4.1 Comparison to Other ASP Systems

Compared systems. We compared DLV^{DB} with state-of-the-art ASP systems, namely DLV [14], Gnt2 [12], ClaspD [8], Smodels [17], and Cmodels [10]. DLV^{DB} and DLV include an internal proprietary grounder, whereas the other systems require an external grounder; we tested both Lparse [19] and GrinGo [9] for this purpose; precisely, given a grounder x and a system y, we run $x|y$ so as to direct the output of x into y; the output of the systems have been directed to null in order to eliminate printing times from the computation of the overall execution times.

It is worth pointing out that all systems but DLV^{DB} and DLV do not support non-ground queries; in order to carry out our tests, we asked these systems to compute all answer sets. However, since tested queries are all non-ground (see below) answer sets must be all computed anyway. Note also that Smodels and GrinGo do not support disjunction; when using these systems we adopted a semantic preserving rewriting, possible for tested queries, which removed disjunctive rules[4]. Finally, the working database of DLV^{DB} was defined on Microsoft SQL Server 2005.

All tests have been carried out on a Pentium IV machine with 500Mb of RAM.

Tested queries. We tested four queries, ranging from simple selections to more complicated ones. Two of these queries have been also used for studying the scalability of tested systems:

- Q_1: select the student IDs and the course descriptions of the examinations they passed (this query involves possible inconsistencies in student IDs, exam records, and course descriptions).
- Q_2: select the first and second names of the professors stored in the database (this query involves possible inclusion dependency violations in relationships involving professors, and possible inconsistencies in exam records).

[4] We used ClaspD also for non disjunctive programs with GrinGo. However, we checked that running times of Clasp are the same as those of ClaspD in these queries.

- Q_3: select pairs of students having at least one common exam (this query involves possible inconsistencies in student IDs and exam records). We leveraged the complexity of this query by filtering out different subsets of exam records.
- Q_4: select pairs of students and course codes of passed examinations such that the professor's first name of the corresponding courses is the same (this query involves possible inconsistencies in student IDs, exam records, and course descriptions). We leveraged the complexity of this query by filtering out different subsets of exam records.

All tested queries are non-ground. We verified also the effectiveness of the unfolding strategy with constants in the queries and obtained very positive results. However, since none of the other tested systems provides query oriented optimizations we do not report here the results obtained in this setting.

It is worth pointing out that all tested queries, even if simple when considered in a single database context, become quite complex when dealing with data integration and inconsistency handling. As an example, query Q_1 on the single global schema can be expressed simply as:

```
Q1(StudID, CDesc):-course(CCode,CDesc),
                exam_record(StudID,CCode,FN,SN,City,Address,Tel,Degree).
```

however, when introducing mappings with the sources and inconsistency handling, the program allowing to answer Q_1 becomes disjunctive (see the example in Section 3). Due to space constraints we can not show here the complete encodings of tested queries. The interested reader can find them in the on-line Appendix at
http://www.mat.unical.it/terracina/rr08/Appendix.pdf.

Results and discussion. Test results are shown in Figure 3. In the graphs, we used the notation x:y to denote the system y coupled with the grounder x; moreover, to simplify the notation, we used symbol L (resp. G) to denote Lparse (resp. GrinGo).

Results of queries Q_1 and Q_2 are shown in Figure 3(a). We can observe that the amount of data involved by these queries is still manageable by all tested systems in main memory. DLV^{DB} and DLV present comparable performances and they are at least 50% faster than other systems. In these queries, there is no substantial difference in using Lparse or GrinGo.

The total amount of data involved by Q_3 and Q_4 is manageable by DLV^{DB} only. We then used different subsets of exam recors to test all the systems.

The scalability of query Q_3 is illustrated in Figure 3(b). Here (and in Figure 3(c)) the line of a system stops when it (or the associated grounder) has not been able to solve the query. Note that no system but DLV^{DB} has been capable of handling 100% of input data, due to lack of memory. Specifically, for this query, grounders were able to complete the computation, but systems not. As for obtained results, it is possible to observe that in this query, when coupled with GrinGo, systems behave generally better than with Lparse, at least for small inputs. Performances of DLV^{DB} are comparable to those of the other systems with Lparse for small inputs, but it behaves much better for bigger data sizes.

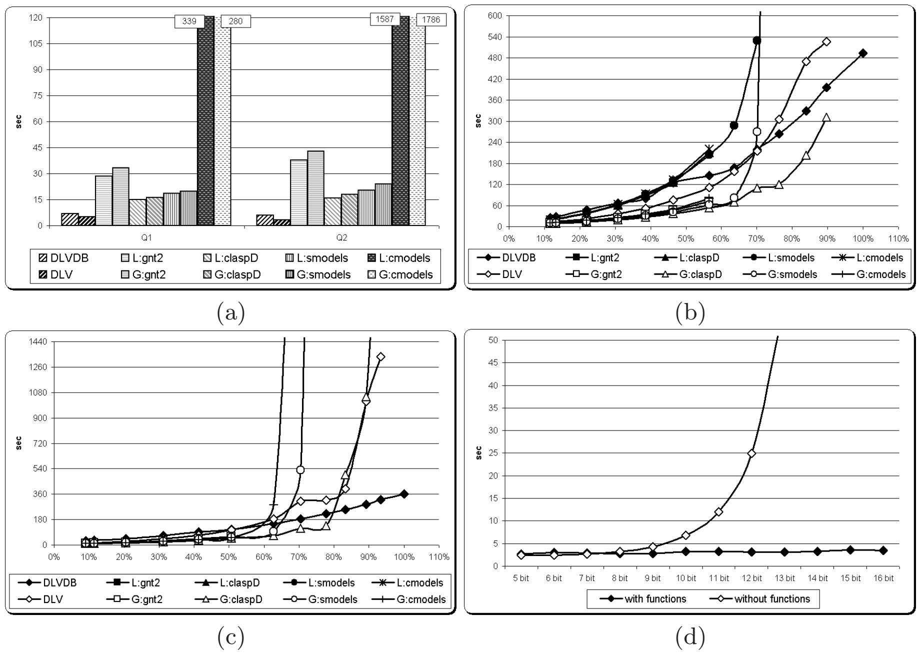


Fig. 3. Results for : (a) Q_1, and Q_2; (b) Q_3; (c) Q_4; (d) Q_5

Notably ClaspD with GrinGo presents the best performance for Q_3 until it is able to handle data in main memory.

Results for query Q_4 are shown in Figure 3(c). Here, Lparse has not been able to complete the grounding in reasonable time even for the smallest data set (we stopped it after 12 hours). Hence, only results with GrinGo are presented (which has been able to complete the grounding for plotted data). Here, again, DLV^{DB} allows handling bigger data sizes than the other systems which, at some point, are subject to memory overflow. Also, the performances of DLV^{DB} in small data sets are extremely competitive.

Finally, Table 1 summarizes the biggest data sets handled by each system for queries Q_3 and Q_4 (the 0% in Q_4 are due to Lparse fault).

Table 1. Biggest data sets handled by tested systems for Q_3 and Q_4

			Lparse:				GrinGo:			
	DLV^{DB}	DLV	Gnt2	ClaspD	Smodels	Cmodels	Gnt2	ClaspD	Smodels	Cmodels
Q_3	100%	90%	46%	57%	70%	57%	57%	90%	84%	57%
Q_4	100%	93%	0%	0%	0%	0%	24%	93%	78%	78%

4.2 Testing Extended Capabilities of the System

Tests on functions. We tested the capability to improve usability and efficiency of DLV^{DB} via functions for a typical real world problem, namely data transformation. Data transformation is particularly relevant also in data integration to uniform data representations among involved sources.

In particular, we considered the problem of transforming integer numbers in their binary representation. This task can be encoded also in datalog (see the on-line Appendix). We then designed a further query, named Q_5, aiming simply at transforming marks stored in input exam records from integers to binaries. We defined two variants of Q_5, one with and one without function calls. In order to measure the scalability of DLV^{DB} with and without functions in this query, we considered the conversion to binary numbers having 5 to 16 bits. Obtained results are shown in Figure 3(d).

The Figure clearly shows the significant advantage of using functions in this context. In fact, the execution time of Q_5 with functions is almost constant because it requires a fixed number of function calls (one for each mark to convert), independently of the number of bits. To the contrary, the standard datalog version must generate all the binary numbers in the admissible range; this explains the exponential growth of the response time.

5 Conclusions

In this paper we presented some enhancements to DLV^{DB} devoted to improve both expressiveness of its supported language and its efficiency. We showed that proposed improvements make DLV^{DB} particularly suited for data intensive applications; moreover we showed that DLV^{DB} exemplifies the usage of DLP for those problems characterized by both declarative and procedural components, via the usage of external function calls. Tests results are very encouraging, showing that DLV^{DB} can handle larger amounts of data in less time than competitor systems.

As for future work we plan to further extend language expressiveness with the capability of handling lists and sets; moreover, we plan to investigate the landscapes of language extensions allowing some controlled forms of side-effects in external functions. It is in fact our opinion that such features may be highly relevant in several contexts, such as intelligent agents interaction.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Specifying and Querying Database Repairs using Logic Programs with Exceptions. In: Proc. of the Fourth International Conference on Flexible Query Answering Systems (FQAS 2000), pp. 27–41 (2000)
2. Beeri, C., Ramakrisnhan, R.: On the power of magic. Journal of Logic Programming 10(1-4), 255–259 (1991)
3. Bennardo, G.: Interrogazione dati basata su ontologie, Master Thesis in Computer Science (2008)
4. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Int. Joint Conference on Artificial Intelligence (IJCAI 2003), pp. 16–21 (2003)

5. Calimeri, F., Cozza, S., Ianni, G.: External sources of knowledge and value invention in logic programming. Annals of Mathematics and Artificial Intelligence 50, 333–361 (2007)
6. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In: Proc. of the 18th Int. Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico, pp. 847–852 (2003)
7. Leone, N., et al.: The infomix system for advanced integration of incomplete and inconsistent data. In: Proc. of 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), Baltimore, Maryland, USA, 2005, pp. 915–917. ACM Press, New York (2005)
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Clasp: A conflict-driven answer set solver. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 260–265. Springer, Heidelberg (2007)
9. Gebser, M., Schaub, T., Thiele, S.: GrinGo: A new grounder for answer set programming. In: Int. Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), Tempe, AZ, USA, pp. 266–271 (2007)
10. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Jornal of Automated Reasoning 36(4), 345–377 (2006)
11. Ianni, G., Martello, A., Panetta, C., Terracina, G.: Efficiently querying RDF(S) ontologies with Answer Set Programming. Journal of Logic and Computation (special issue) (forthcoming)
12. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.: Unfolding partiality and disjunctions in stable model semantics. TOCL 7(1), 1–37 (2006)
13. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. PODS 2002, pp. 233–246 (2002)
14. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Trans. Comput. Log. 7(3), 499–562 (2006)
15. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Information and Computation 135(2), 69–112 (1997)
16. Mumick, I.S., Finkelstein, S.J., Pirahesh, H., Ramakrishnan, R.: Magic conditions. ACM Trans. Database Systems 21(1), 107–155 (1996)
17. Niemelä, I., Simons, P., Syrjänen, T.: Smodels: A System for Answer Set Programming. In: Proc. of the 8th Int. Workshop on Non-Monotonic Reasoning (NMR 2000), Colorado, USA (April 2000)
18. Ricca, F., Leone, N.: Disjunctive Logic Programming with types and objects: The DLV$^+$ System. Journal of Apllied Logic 5(3), 545–573 (2007)
19. Syrjänen, T.: Lparse 1.0 user's manual (2002), http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz
20. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theory and Practice of Logic Programming (TPLP) 8(2), 129–165 (2008)

A Semantic Web Reasoner for Rules, Equations and Constraints

Daniel Elenius, Grit Denker, and Mark-Oliver Stehr

SRI International, Menlo Park, California, USA
`firstname.lastname@sri.com`

Abstract. We describe a reasoner for OWL ontologies and SWRL policies used on cognitive radios to control dynamic spectrum access. In addition to rules and ontologies, the reasoner needs to handle user-defined operations (e.g., temporal and geospatial). Furthermore, the reasoner must perform sophisticated constraint simplification because any unresolved constraints can be used by a cognitive radio to plan and reason about its spectrum usage. No existing reasoner supported all these features. However, the term rewriting engine Maude, augmented with narrowing, provides a promising reasoning mechanism. This allows for a behavior similar to that of a logic programming system, while constraint simplification rules as well as operations can easily be defined and processed. Our system and general approach will be useful for other problems that need sophisticated constraint processing in addition to rule-based reasoning, or where new operations need to be added. The implementation is efficient enough to run on resource-constrained embedded systems such as software-defined radios.

1 Introduction

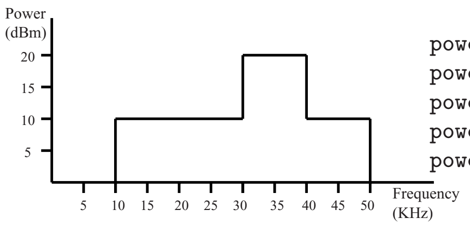
The radio frequency spectrum is a finite resource, and demand for it is increasing. Large, robust, and agile radio networks are very difficult to achieve with traditional methods. *Cognitive radios* and *dynamic spectrum access* offers a solution, where radios can use sensors to avoid interference and *reason* about spectrum usage. This area offers an interesting application domain for Semantic Web technologies.

In DARPA's neXt Generation (XG) program, declarative *policies* are used to control access to the spectrum resource. Policies define circumstances under which radios are allowed to transmit, in terms of frequencies used, power levels, geographic location, time, and so on. These concepts are defined in *ontologies.* A *policy reasoner* is used to decide whether a transmission is allowed. More background can be found in [1].

The ontologies and policies are encoded in OWL and SWRL. The semantics is the usual first-order model theory. However, the reasoning problem is not a straightforward Description Logic subsumption problem. This paper describes our policy reasoner, which is a general-purpose reasoner for OWL and SWRL, but with several additional features. Our intent here is not to give a formal characterization of our reasoner, but to motivate and describe the system, and to contrast it with other reasoning technologies.

Our approach is to translate to a target language that has reasoning mechanisms appropriate to our domain. There are several desired features of such a language and its reasoning system. Policies can be permissive or restrictive. It is intuitive to write policies as *rules*, e.g., of the form `Allow` $\Leftarrow$ *Constraints* (for permissive policies), where we try to prove `Allow` by solving the constraints. (Replace `Allow` by `Disallow` for restrictive policies). It is also useful to define auxiliary predicates using rules, for modularity, reusability, and convenience of specification. This speaks in favor of a logic programming type of language.

We also need to define certain operations, such as arithmetic on time primitives, and calculation of the distance between two geographic points. Pure logic programming does not provide an appropriate computation framework for this, and because we want purely declarative policies, procedural attachments are not an option. However, these types of operations can be defined in a natural, declarative way that allows for efficient computation, using *functions* and *equational specifications*. Another use of functions is to define so-called power masks, i.e., functions from frequency to power levels (see Figure 1).



$$\begin{aligned}
\texttt{power}(\texttt{f}) &= 0.0 \Leftarrow \texttt{f} < 10.0\\
\texttt{power}(\texttt{f}) &= 10.0 \Leftarrow 10.0 \leq \texttt{f} \wedge \texttt{f} < 30.0\\
\texttt{power}(\texttt{f}) &= 20.0 \Leftarrow 30.0 \leq \texttt{f} \wedge \texttt{f} < 40.0\\
\texttt{power}(\texttt{f}) &= 10.0 \Leftarrow 40.0 \leq \texttt{f} \wedge \texttt{f} < 50.0\\
\texttt{power}(\texttt{f}) &= 0.0 \Leftarrow 50.0 \leq \texttt{f}
\end{aligned}$$

Fig. 1. A power mask (left) is a function from frequency to power. Such functions can be intuitively defined using conditional equations (right).

Another ubiquitous feature in spectrum policies are *numerical constraints*, for frequency ranges, power levels, minimum distance to other transmitters, and so on. Interesting problems arise from the combination of numerical and other logical constraints.

To summarize, we need a reasoner that allows rules in the logic programming style, functions defined using equations, and flexible handling of numerical constraints.

This remainder of this paper is organized as follows. In Section 2 we examine existing reasoning technologies with regard to the desired features. Section 3 describes the system in which we implemented our reasoner, and its underlying logic. Section 4 describes our reasoner, and shows how it can be used as a Web reasoning tool. We end with some conclusions in Section 5.

2 Rules, Equations, and Constraints

Here, we examine the features needed by the policy language in more detail, and discuss existing technologies supporting them. There are two somewhat (but not completely) separate issues. The first is the combination of logic programming with functions and equations. The second is the addition of a flexible notion of constraints.

2.1 Combining Rules and Equations

Combining a rule system with functions and equations could be seen as combining the logic programming and functional programming paradigms. There is a large body of work on this topic (see [2,3] for extensive surveys). Following [4], we take the relational paradigm to be about *multidirectional*, *nondeterministic*, and *not necessarily convergent* computation, and the functional paradigm to be about *directional*, *deterministic*, and *convergent* computation (see Table 1).

Table 1. Comparison of two paradigms of computation

Functions, Equations	Relations, Rules
Deterministic *No failure or backtracking*	Nondeterministic *Failure/backtracking*
Directional *Takes inputs and produces outputs*	Multidirectional *No specific inputs/outputs*
Terminating *Usually expected to terminate on legal input*	Not necessarily terminating *Can enumerate infinitely many instances of arguments*
Evaluation *Reduction*	Deduction *Search/resolution/ unification*

Having a language that supports only one of the two paradigms usually forces users to make unnatural encodings in order to support the missing functionality. One one hand, functions encoded as relations in logic programming cannot take advantage of the efficient evaluation of deterministic functions that a functional programming language can perform. On the other hand, functional programming cannot take advantage of the built-in search and partially instantiated data structures that logical programming supports. Thus, a combination of both paradigms is called for.

There are two fundamental approaches to the problem of combining relations and functions [3]. One is to start with the relational approach and add functions. The other is to start with the functional approach and add relations.

In some sense, relations are more general than functions and can emulate functional behavior. For example, functional notations can be translated into Prolog through *flattening*, and determinism of function evaluation can be achieved by using Prolog features like cuts. This approach is used in [5] and [6], which both take logic programming as the starting point, and add functional notation through a translation approach. These approaches make a syntactical distinction between rules and equations, and between relations and functions. Normal Prolog mechanisms are used for rules and relations, and the translated versions with a special encoding to behave like functions are used for the equational/functional side.

Systems that take functional programming as the starting point instead treat relations and logical connectives (*and*, *or*, etc.) as boolean functions, and generalize their expressiveness by allowing new variables on the right-hand sides of equations. Thus, these languages typically do not make a distinction between relations and functions.

Regardless of the starting point, the choices of operational principles are similar, the main choices being *residuation* (e.g., Le Fun [4]) or *narrowing* (e.g., Curry [7]). The idea of residuation is to delay evaluation of terms containing logical variables until the variables have been instantiated due to the evaluation of other terms. Narrowing works by using unification instead of matching when a redex is matched with the left-hand side of an equation, thus allowing logical variables in the redex. For example, given the equations

$$brother(Phil) = Tom$$

$$brother(Phil) = Jack$$

and the redex $brother(x) = Tom$, normal reduction does not work since we have a variable in the redex. However, with narrowing, the redex unifies with the first equation and the binding $x = Phil$. In general, narrowing can result in several different solutions, which means that backtracking or some equivalent mechanism must be provided. For example, the redex $brother(Phil) = y$ unifies with both equations, with the bindings $y = Tom$ and $y = Jack$. Narrowing is the most general approach, encompassing both unification and reduction, and thus supporting both the functional and the relational paradigms. Our solution is based on a functional language, Maude [8], and uses narrowing.

2.2 Constraints

Constraints are fundamental in the policies we considered in the XG project – in fact we take the view that policies *are* constraints. There are many different notions of "constraints". We will make clear what we mean by constraints, and how this compares to the constraints of Constraint Logic Programming (CLP) [9].

In the XG architecture, the policy reasoner must prove a special atom `Permit` based on the policies and facts available:

$$\mathit{Facts}, \mathit{Policies} \vdash \texttt{Permit}$$

Policies will have axioms about the `Permit` predicate, so that the proof obligation becomes

$$\mathit{Facts} \vdash \mathit{Constraint}$$

where *Constraint* is a formula resulting from combining all those `Permit` axioms. The *facts* come from a transmission request, where the radio states what its current configuration is, what its intended transmission looks like, and what the state of the environment is, as far as the radio can determine by using its sensors. As a simplistic example, consider the following policies:

$$\texttt{Permit} \Leftrightarrow \texttt{Allow} \wedge \neg\texttt{Disallow}$$

$$\texttt{Allow} \Leftarrow 500 < \texttt{freq} \wedge \texttt{freq} < 1000$$

$$\texttt{Allow} \Leftarrow 1200 < \texttt{freq} \wedge \texttt{freq} < 1400$$

$$\texttt{Disallow} \Leftarrow 900 < \texttt{freq} \wedge \texttt{freq} < 1300$$

The first, “top-level”, policy relates permissive and restrictive policies. The given top-level policy just says that we need to find *some* policy that allows, and *no* policy can disallow. Other top-level rules can be used to account for priorities and other relationships between policies. With these policies, after expanding the definition of `Permit`, we get the proof obligation

$$\begin{aligned} \mathit{Facts} \vdash (500 < \texttt{freq} \wedge \texttt{freq} < 1000 \vee 1200 < \texttt{freq} \wedge \texttt{freq} < 1400) \\ \wedge\neg(900 < \texttt{freq} \wedge \texttt{freq} < 1300) \end{aligned}$$

which can be further simplified to

$$\mathit{Facts} \vdash 500 < \texttt{freq} \wedge \texttt{freq} \leq 900 \vee 1300 \leq \texttt{freq} \wedge \texttt{freq} < 1400$$

If *Facts* contains for instance $\texttt{freq} = \texttt{800}$, the whole constraint reduces to `True`, which means that the proof is completed and the radio can transmit. However, we are also interested in the case where such facts are *not* provided, because radios can make underspecified requests as a way of querying for available transmission opportunities. Thus, whatever remains of the constraint after simplification has been performed should be returned as a result to the radio.

The description above could be subsumed under a very general view of CLP. However, the variant of CLP that has been implemented in current Prolog systems is less general, in at least three respects. First, CLP does not handle negation in a clean, logical way. For example, we would like to be able to get simplifications like

$$\neg\texttt{freq} < 500 \rightarrow \texttt{freq} \geq 500$$

In Prolog, negation is handled by the negation-as-failure method, which precludes such inferences. Second, CLP does not handle disjunction. For example, it cannot perform the simplification

$$\texttt{freq} < 500 \vee \texttt{freq} > 400 \rightarrow \texttt{True}$$

Third, in Prolog/CLP, only special constraint formulas are returned when they are not completely satisfied, whereas we view *all* formulas as constraints. While there are proposals to handle the negation [10] and disjunction [11] limitations, the third limitation is of a more fundamental nature.

The view of constraints and constraint simplification that we have suggested above lends itself very well to a formalization and implementation as a term rewriting system [12], where the derivations above are instances of some rewrite rules of the form $A \to B$. Indeed, at first glance it may look as if most functional programming languages (e.g., Haskell or ML) or functional logic programming languages (e.g., Curry or Escher) can be used for this purpose, since they allow us to write equations, which are usually interpreted as left-to-right rewrite rules. However, two limitations of these languages prohibit this: their *constructor discipline* and their inability to handle *associative* or *commutative* functions.

Constructors are a subset of the function symbols that cannot be further reduced. Most functional programming languages restrict the equations one can write such that the left-hand sides must be of the form $f(t_1, \ldots, t_n)$, where f is a nonconstructor function symbol and t_i are terms containing only constructor symbols or variables. This constructor discipline allows one to define computable functions and to execute them in an efficient way, but it limits us when we want to define a more general rewrite relation, such as our constraint simplification relation. For example, we will need rewrite rules/equations such as

$$A \wedge \texttt{true} = A$$

$$A \vee \texttt{true} = \texttt{true}$$

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

where A, B, and C are boolean variables. From the first two equations, it is clear that neither $\wedge$ nor $\vee$ are constructors, as they can be eliminated (in fact, `true` and `false` are the only constructors of the boolean algebra). Thus, the third equation does not adhere to the constructor discipline, since it contains nested nonconstructor function symbols.

To show the deficiency of functional programming languages with regard to *associative* and *commutative* functions, consider what would happen if we try to reduce a term $\texttt{true} \vee A$ using the rules above. We want to use the second rule, but it does not match, since the terms are in the wrong order. However, $\vee$ is commutative, and we could encode this using another equation,

$$A \wedge B = B \wedge A$$

but adding this equation will make the term rewriting system nonterminating, since any term rewritten by this equation still matches the equation, and can thus be rewritten again indefinitely. A similar argument applies to the associativity of conjunction and other operators. One solution is *AC matching*, where we can declare that an operator is associative and/or commutative, instead of using an equation. AC matching means matching modulo these properties, and can be done in a built-in, terminating way. Thus, our solution is to use a system which does not demand a constructor discipline and which supports AC matching, namely, Maude [8].

3 Equational Logic, Rewriting Logic, and Maude

Equational Logic. (EL) [13] is the subset of first-order logic with $=$ as the only predicate symbol, and equations as the only formulas (i.e., there are no logical connectives). The rules of deduction of EL are *reflexivity*, *congruence*, *transitivity*, and *symmetry*. Despite being a very small subset of first-order logic, equational logic can be used to define any computable function. Furthermore, EL can be used as a programming language, by treating equations as left-to-right rewrite rules (i.e., ignoring the symmetry rule), and using *reduction* as the operational semantics. Viewed as rewrite systems, theories in EL are expected to be terminating and confluent. This means that the order in which redexes and rewrite rules are chosen does not matter; we will reach the same result regardless, and in a finite number of steps. Sometimes *conditional* equations are allowed, which means that Horn clauses can be used in addition to plain equations, as in the power mask definition in Figure 1.

Rewriting Logic. (RL) [14] is similar to EL on the surface, in that it allows for (possibly conditional) rewrite rules. However, the rules are not semantically equations, and the rule systems are not expected to be confluent. Therefore, reduction cannot be used as the operational semantics, since different choices of redexes and rules can lead to different results. Instead, the rewrite rules are interpreted as nondeterministic state transitions, and the operational mechanism is *search* in the state space. Analogously to EL, RL can also support conditional rewrite rules.

Maude. [8] is a multiparadigm executable specification language encompassing both EL and RL. The Maude interpreter is very efficient, allowing prototyping of quite complex test cases. Maude also provides efficient built-in search and model checking capabilities. Maude is reflective [15], providing a meta-level module that reflects both the syntax and semantics of Maude. Using reflection, the user can program special-purpose execution and search strategies, module transformations, analyses, and user interfaces. Maude sources, executables for several platforms, the manual, a primer, cases studies, and papers are available from the Maude website `http://maude.cs.uiuc.edu`.

We briefly summarize the syntax of Maude that is used in this paper. Maude has a module system, with

– *functional* modules, specifying equational theories, which are declared with the syntax `fmod...endfm`
– *system* modules, which are rewrite theories specifying systems of state transitions; they are declared with the syntax `mod...endm`

These modules have an *initial model semantics* [16]. Immediately after the module's keyword, the *name* of the module is given. After this, a list of imported submodules can be added. One can also declare *sorts* and *subsorts* and *operators*. Operators are introduced with the `op` keyword followed by the operator name, the argument and result sorts. An operator may have mixfix syntax, with the name containing '`_`'s marking the argument positions. A binary operator may be declared with equational *attributes*, such as `assoc`, `comm`, and `id: <identity element>` stating, for example, that the operator is associative, commutative, and specifying an identity element for the operation. Such attributes are then used by the Maude engine to match terms *modulo* the declared axioms. Equational axioms are introduced with the keyword `eq` (or `ceq` for conditional equations) followed by the two terms being declared equal separated by the equality sign `=`. Rewrite rules are introduced with the keyword `rl` (or `crl` for conditional rules) followed by an optional rule label, and terms corresponding to the premises and conclusion of the rule separated by the rewrite sign `=>`. Variables appearing in axioms and rules (and commands), may be declared globally using the keyword `var` or `vars`, or "inline" using the variable name and its sort separated by a colon; for example, `n:Nat` is a variable named `n` of sort `Nat`. Rewrite rules are not allowed in functional modules.

Maude has a `reduce` command for equational reduction in functional modules, and a `search` command for breadth-first search in the state space of system modules. The search mechanism allows searching for the first answer, all answers, or only answers matching some goal term. The search mechanism encompasses the reduction mechanism, as equational reduction is performed before each application of rewrite rules.

3.1 Logic Programming in Maude

Maude also has a `narrow` command. Narrowing in Maude is similar in many ways to the `search` mechanism mentioned above. Like search, narrowing nondeterministically selects rewrite rules, generates choice points, and can return answers in the same ways. There are two differences: 1) new variables are allowed on the right-hand side of rewrite rules, and 2) when there are uninstantiated variables in a redex, unification is used instead of matching.

As mentioned in Section 2.1, the addition of narrowing to a functional language gives us the ability to subsume logic programming. We encode relations and logical connectives as boolean functions. The logical connectives are defined in a `BOOL` module in Maude's "prelude", which contains statements like

```
op _and_ : Bool Bool -> Bool [assoc comm prec 55] .
op _or_ : Bool Bool -> Bool [assoc comm prec 59] .
op not_ : Bool -> Bool [prec 53] .
vars A B C : Bool .
eq true and A = A .
eq false and A = false .
eq A and A = A .
```

The `assoc` and `comm` attributes declare the associativity and commutativity of the operators. The `prec` attribute sets the precedence of an operator so that the mixfix notation is parsed correctly without a proliferation of parentheses. Note that `not` is simply a truth function that takes `true` to `false` and vice versa, and corresponds to classical negation. We do not have a notion of negation-as-failure since we deal with a language with classical first-order models.

User-defined predicates are encoded in a way similar to the connectives. For example, an n-ary predicate `P` is encoded as

```
op P : T1 ... Tn -> Bool .
```

where `Ti` are the sorts of the arguments of `P`.

Rules are encoded as rewrite rules. For example,

$$\forall x, y, z : uncle(x, y) \Leftarrow parent(x, z) \wedge brother(z, y)$$

is encoded in Maude as

```
vars x,y,z : Ind .
rl uncle(x,y) => father(x,z) and brother(z,y).
```

Note the implicit existential variable `z` on the right-hand side. Normal rewriting logic does not support this, but narrowing allows us to handle this. We do not allow existential variables in a negative context (i.e., under an odd number of `not`s). This would correspond to universal quantification, which the system does not support.

Facts are rules without bodies. For example, *father*(*John*, *Bob*) is encoded as

```
rl father(John,Bob) => true .
```

Note the use of rewrite rules (`rl`) rather than equations (`eq`) for user-defined facts and rules. This is motivated by operational concerns; we want to be able to use narrowing on these facts and rules. The model-theoretic semantics of Maude rewrite rules as state transitions does not directly reflect our interpretation of the rules as implications in first-order logic. However, our use of Maude is sound with regard to the first-order model theory.

4 Maude as a Web Reasoning Tool

We now describe how Maude can be used as a Web reasoning tool. First we describe how to translate (parts of) OWL ontologies and SWRL rules into Maude, and how the system can be used in a way similar to other rule-based systems like Prolog. Then we describe the novel features where our system goes beyond other rule-based systems: user-defined or "built-in" operations, and domain-specific simplification rules. Our examples come from the spectrum policy domain as discussed above, but we believe that these features are generally useful for a wide variety of problems. Finally, we discuss some implementation details.

4.1 Encoding OWL and SWRL in Maude

To use Maude as the reasoning engine for spectrum policies, we need to translate our policies to Maude. The policies are written as SWRL rules, i.e., Horn clauses, and refer to OWL ontologies. We can also translate a significant portion of axioms from OWL ontologies into Horn logic [17]. Once we have all our statements in Horn clause form, it is straightforward to encode them in a very direct way in Maude, using the scheme described in Section 3.1. Some specifics of the encoding follow.

First, we note that OWL does not have types or sorts in the sense of Maude or other programming languages. However, Maude operators need to be declared with sorted signatures. We therefore introduce one sort of OWL individuals, `Ind`, and one sort of OWL data values, `Data`. These are the two semantic domains of

OWL models. We translate OWL individuals, classes, and properties as follows (`***` denotes a comment in Maude):

```
op Radio1 : -> Ind .                *** an individual
op Radio : Ind -> Bool .            *** a class
op detector : Ind Ind -> Bool .     *** an object property
op frequency : Ind Data -> Bool . *** a datatype property
```

The signatures here are perhaps best understood as follows: An individual is an operator with no argument that returns itself (a constant). A class is an operator that takes an individual as an argument, and returns `true` or `false`, depending on whether or not the individual is a member of the class. A property is an operator that takes a subject and an object as arguments, and returns `true` or `false` depending on whether or not that subject has that object as a property value for the property in question.

We treat *functional* properties separately, translating them as

```
op role : Ind -> Ind             *** a functional object property op
power : Ind -> Data              *** a functional datatype property
```

where the operator works as a function—it takes the subject as an argument and returns the object. This encoding makes reasoning more efficient in Maude, since it is essentially a functional language.

Facts are translated into Maude rewrite rules as in the following examples:

```
rl Radio(Radio1) => true .                  *** class-instance fact
rl detector(Radio1,Detector1) => true .     *** property-value fact
rl role(Radio1) => BaseStation .    *** functional property-value fact
```

Finally, *rules* are translated exactly as shown in Section 3.1.

4.2 Operations

Recall from Section 1 that one of our motivations for using a reasoning technology that supports functions and equations was that we needed to define certain operations on temporal and geospatial entities. These operations should be deterministic, directional, and terminating, and should be evaluated rather than "reasoned" about. In other words, they should be defined as functions. Whereas we need to use rewrite rules (`rl`) for user-defined axioms in order to support more general reasoning using narrowing, we use equations (`eq`) for these defined operations.

Note that SWRL has a number of built-in operations of this functional flavor, e.g., for math, time, comparisons, and strings.[1] We propose that additional operations should be treated in the same way as SWRL built-ins, i.e., used in "built-in atoms" in SWRL rules.

These kinds of operations cannot be defined in OWL or SWRL. The Protégé ontology development environment [18] supports adding implementations of new

[1] See `http://www.w3.org/Submission/SWRL/`

built-ins in Java [19]. However, using a functional framework, as we propose, provides many well-known advantages of declarative languages, such as clear semantics, intuitive and transparent specifications, and better integration with reasoning. In fact, one could give the SWRL builtins a formal semantics (which they currently lack) by defining them using Maude equations.

Ideally, we would like to have an overall semantic framework that encompasses both OWL (and SWRL) and the equational specification of these “built-ins.” This is an area of future research. It is also an open question whether all the operations should be considered “built in” to the reasoner, or whether users could also define their own operations. The latter option would provide great flexibility, as it allows the same reasoner to be applied to new domains, where any new operations required can be defined by users themselves, rather than having to modify the reasoner with new “built-ins”. There are, however, some practical details to resolve, e.g., regarding which syntax to use for these user-defined operations.

Currently, our reasoner specification includes many of the SWRL built-ins, and certain geospatial operations. Because of space restrictions, we cannot show sample implementations of operations here, but our reasoner specification can be downloaded at `http://xg.csl.sri.com/`. See, in particular, the `TIME` and `GEO` modules.

4.3 Simplification Rules

The second motivation for using a reasoning technology that supports equations is our need to write custom constraint simplification rules, as discussed in Section 2.2. In fact, this is where Maude really differentiates itself from most other systems, because it can do reduction, rewriting, and narrowing *modulo associativity and commutativity (AC)*. Handling associativity and commutativity in a built-in way is critical for encoding logical rewriting systems such as the one in question here, because lacking this capability, we would be forced to add AC axioms, which would make the system nonterminating, as discussed previously.

While Maude's prelude already includes certain constraint simplification rules, such as the trivial ones we showed in Section 3.1, we needed to go further and implement our custom rules, for two reasons: 1) the combination of ordering constraints (e.g., $<$, $\leq$) and boolean constraints (e.g., $\wedge$, $\vee$) introduces many opportunities for simplification, and 2) the desire to get answers in a certain form.

The latter point could use some additional clarification. All the simplification rules transform a constraint into another, equivalent, constraint. The purpose of all rules is to move the constraint to a “simpler” form, until we have reached the “simplest” possible form. However, it is not always obvious what the simplest form is. In the radio policy domain, we have some guidance from the domain. The policy engine should return a constraint that can be recognized by the radio as a set of transmission opportunities, where each opportunity is straightforward to interpret. We can achieve this by using *disjunctive normal form* (DNF) as the target of our constraint reduction. A DNF formula is a disjunction of conjunctions, i.e., it has the form

$$(A \wedge B \wedge C) \vee (D \wedge E) \vee (F \wedge G) \vee ...$$

We can think of each of the conjuncts $(A \wedge B \wedge C)$, $(D \wedge E)$, or $(F \wedge G)$ as an opportunity, because it is enough for the radio to satisfy *one* of the disjunctions in order to satisfy the entire constraint. For example, if the radio provides the facts D and E, then the formula above reduces to

$$(A \wedge B \wedge C) \vee \texttt{True} \vee (F \wedge G) \vee ... \rightarrow \texttt{True}$$

using the rule

$$\texttt{True} \vee P \rightarrow \texttt{True}$$

Furthermore, for the chosen opportunity, all the constraints have to be satisfied.

Simplifying to DNF, however, is not enough. For example, the following constraint is in DNF

$$(\texttt{freq} < 500 \wedge \texttt{power} < 10) \vee (\texttt{freq} > 400)$$

but it can be simplified further to

$$\texttt{power} < 10 \vee \texttt{freq} > 400$$

This constraint is still in DNF and equivalent to, but simpler in some sense than, the first form. Our simplification rules take care of such cases.

4.4 Implementation

Above, we have discussed the *principles* of our engine. Here, we describe the main components of our implementation. The engine needed to be implemented in C/C++ in order to run on resource-constrained radios. Maude is implemented in C++ and highly optimized, so this posed no particular problem. The policies are written in OWL, SWRL, and SWRL FOL, using the XML presentation syntax. There was no existing C/C++ parser for OWL, and no software support for SWRL FOL. Thus, we implemented our own parser/writer for OWL+SWRL+SWRL FOL using the XML presentation syntax.

The Maude reasoner back end consists of several distinct parts, as shown in Figure 2:

– A translator from the parser's representation of OWL/SWRL to Maude, using the encoding in Section 4.1.
– The Maude reasoner specification, containing Maude equations representing the simplification rules and built-in operations.
– The Maude engine itself. Normally, Maude runs as an interactive console-based tool, but we used an experimental programming API to Maude.
– A translator from Maude results back to OWL/SWRL.

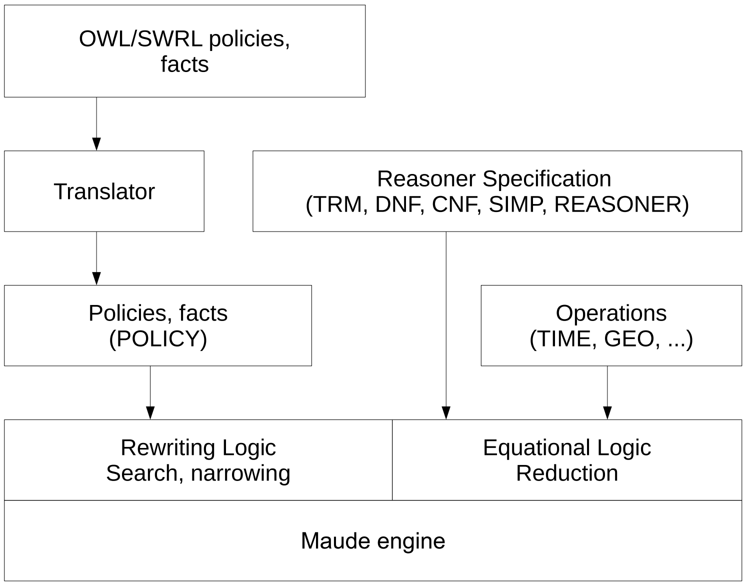

OWL/SWRL policies, facts
Translator
Reasoner Specification (TRM, DNF, CNF, SIMP, REASONER)
Policies, facts (POLICY)
Operations (TIME, GEO, ...)
Rewriting Logic Search, narrowing
Equational Logic Reduction
Maude engine

Fig. 2. Components of the Maude-based reasoner

On a more detailed level, the Maude specification consists of a number of Maude modules: `TRM`, `TIME`, `GEO`, `POLICY`, `SIMP`, `DNF`, `CNF`, and `REASONER`. The `TRM` module contains basic definitions of the boolean algebra, arithmetic, and ordering constraints. The `TIME` and `GEO` modules contain built-in functions for temporal and geospatial reasoning. The `POLICY` module contains the translation of all the policies and facts that are currently loaded. When the reasoner first starts, the `POLICY` module is empty. Whenever new policies or facts are loaded, this module is updated. `SIMP`, `DNF`, and `CNF` contain different parts of the proof system. `SIMP` does a number of simplifications such as eliminating negation (as far as possible) and implication. `CNF` converts to conjunctive normal form, and does some simplifications that can be done only in this form. `DNF` converts to disjunctive normal form, and does some simplifications that can be done only in this form. If these three modules were combined into one, the reasoning would never terminate. For example, it could transform between `CNF` and `DNF` back and forth indefinitely. The `REASONER` module controls the ordered execution of the reasoning modules by using the Maude meta level. First, narrowing is done in the `POLICY` module, then reduction in `SIMP`, `CNF`, and `DNF`, in that order. The narrowing step looks for *all* answers. To exploit simplification opportunities between different answers, the answers are `or`'d together before simplification, again using meta level Maude code.

5 Conclusions

Dynamic spectrum access offers an interesting application area for Semantic Web technologies. OWL allows us to define concepts related to the radio domain, and spectrum access can be controlled by policies written as SWRL rules. In addition to rules and ontologies, we needed to define operations (e.g., temporal and geospatial) using equational specifications. Furthermore, we wanted the reasoner to perform sophisticated *constraint simplification*. Any unresolved constraints can be used by a cognitive radio to plan and reason about its spectrum usage.

No existing reasoner supported all these features, but we found that Rewriting Logic provided a promising reasoning mechanism. We built our reasoner on top of Maude, an Equational Logic and Rewriting Logic system developed at SRI. Maude was extended with *narrowing*, which allows it to achieve behavior similar to that of a logic programming system, i.e., goal-oriented reasoning with rules. At the same time, the equational part of Maude is ideal for defining a constraint simplification system, as well as for defining operations. Thus, we were able to include all our desired features in one system. The resulting system subsumes both equational logic and logic programming. It can also easily be extended or modified in two ways. First, because the constraint simplification part is specified in the Maude language, as opposed to hard coded into the reasoning engine, it can be modified to better suit different needs. Second, new operations can be added as additional equational specifications.

The policy domain motivated our work, but the reasoner is not limited to this domain, since it operates on any OWL ontologies and SWRL rules. In particular, our system will be useful for problems that need sophisticated constraint processing in addition to rule-based reasoning, or where new operations need to be added. The implementation is also efficient enough to run on resource-constrained embedded systems such as software-defined radios.

Acknowledgments

We thank Steven Eker (SRI) for maintaining the Maude system and responding to our feature requests, including the entire narrowing functionality. This research was supported by DARPA's neXt Generation (XG) Communications Program under Contract Number FA8750-05-C-0230.

References

1. Wilkins, D.E., Denker, G., Stehr, M.-O., Elenius, D., Senanayake, R., Talcott, C.: Policy-based cognitive radios. IEEE Wireless Communications 14(4), 41–46 (2007)
2. Hanus, M.: The integration of functions into logic programming: From theory to practice. J. Log. Program 19(20), 583–628 (1994)
3. Hanus, M.: Multi-paradigm declarative languages. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 45–75. Springer, Heidelberg (2007)
4. Ait-Kaci, H., Lincoln, P., Nasr, R.: Le fun: Logic, equations, and functions. In: IEEE Symposium on Logic Programming (1987)
5. Casas, A., Cabeza, D., Hermenegildo, M.V.: A syntactic approach to combining functional notation, lazy evaluation, and higher-order in LP systems. In: Hagiya, M., Wadler, P. (eds.) FLOPS 2006. LNCS, vol. 3945, pp. 146–162. Springer, Heidelberg (2006)

6. Naish, L.: Adding equations to NU-Prolog. In: Małuszyński, J., Wirsing, M. (eds.) PLILP 1991. LNCS, vol. 528, pp. 15–26. Springer, Heidelberg (1991)
7. Hanus, M., Kuchen, H., Moreno-Navarro, J.: Curry: A truly functional logic language. In: Proc. ILPS 1995 Workshop on Visions for the Future of Logic Programming (1995)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. (eds.): All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
9. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. Journal of Logic Programming 19(20), 503–581 (1994)
10. Stuckey, P.J.: Negation and constraint logic programming. Information and Computation 118, 12–33 (1995)
11. Backer, B.D., Beringer, H.: A CLP language handling disjunctions of linear constraints. In: ICLP, pp. 550–563 (1993)
12. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York (1998)
13. O'Donnell, M.J.: Equational logic as a programming language. Massachusetts Institute of Technology, Cambridge, MA, USA (1985)
14. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96, 73–155 (1992)
15. Clavel, M., Meseguer, J.: Reflection and strategies in rewriting logic. In: Rewriting Logic Workshop 1996. Electronic Notes in Theoretical Computer Science, number 4. Elsevier, Amsterdam (1996), http://www.elsevier.nl/locate/entcs/volume4.html
16. Wirsing, M.: Algebraic specification. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 675–788. North-Holland, Amsterdam (1990)
17. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proc. Twelfth International World Wide Web Conference (WWW 2003), pp. 48–57. ACM, New York (2003)
18. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé OWL plugin: An open developoment environment for Semantic Web applications. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 229–243. Springer, Heidelberg (2004)
19. O'Connor, M., Das, A.: A mechanism to define and execute SWRL built-ins in Protégé-OWL (2006)

What Is Approximate Reasoning?*

Sebastian Rudolph[1], Tuvshintur Tserendorj[2], and Pascal Hitzler[1]

[1] AIFB, University of Karlsruhe, Germany
[2] FZI Karlsruhe, Germany

Abstract. Approximate reasoning for the Semantic Web is based on the idea of sacrificing soundness or completeness for a significant speed-up of reasoning. This is to be done in such a way that the number of introduced mistakes is at least outweighed by the obtained speed-up. When pursuing such approximate reasoning approaches, however, it is important to be critical not only about appropriate application domains, but also about the quality of the resulting approximate reasoning procedures. With different approximate reasoning algorithms discussed and developed in the literature, it needs to be clarified how these approaches can be compared, i.e. what it means that one approximate reasoning approach is better than some other. In this paper, we will formally define such a foundation for approximate reasoning research. We will clarify – by means of notions from statistics – how different approximate algorithms can be compared, and ground the most fundamental notions in the field formally. We will also exemplify what a corresponding statistical comparison of algorithms would look like.

1 Introduction

In different application areas of Semantic Technologies, the requirements for reasoning services may be quite distinct; while in certain fields (as in safety-critical technical descriptions) soundness and completeness are to be rated as crucial constraints, in other fields less precise answers could be acceptable if this would result in a faster response behaviour.

Introducing approximate reasoning in the Semantic Web field is motivated by the following observation: most nowadays' specification languages for ontologies are quite expressive, reasoning tasks are supposed to be very costly with respect to time and other resources – this being a crucial problem in the presence of large-scale data. As a prominent example, note that reasoning in most description logics which include general concept inclusion axioms (which is simply standard today, and e.g. the case in OWL DL) is at least EXPTIME complete, and if nominals are involved (as for OWL DL) even NEXPTIME complete. Although

* Research reported in this paper was partially supported by the EU in the IST project NeOn (IST-2006-027595, http://www.neon-project.org/)), by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project, and by the German Federal Ministry of Education and Research (BMBF) under the Theseus project, http://theseus-programm.de.

those worst case time complexities are not likely to be thoroughly relevant for the average behaviour on real-life problems, this indicates that not every specifiable problem can be solved with moderate effort.

In many cases, however, the time costs will be the most critical ones, as a user will not be willing to wait arbitrarily long for an answer. More likely, she would be prone to accept "controlled inaccuracies" as a tradeoff for quicker response behaviour. However, the current standard reasoning tools (though highly optimized for accurate, i.e., sound and complete reasoning) do not comply with this kind of approach: in an all-or-nothing manner, they provide the whole answer to the problem after the complete computation. It would be desirable, however, to have reasoning systems at hand which can generate good approximate answers in less time, or even provide "anytime behaviour", which means the capability of yielding approximate answers to reasoning queries during ongoing computation: as time proceeds, the answer will be continuously refined to a more and more accurate state until finally the precise result is reached. Clearly, one has to define this kind of behaviour (and especially the notion of the intermediate inaccuracy) more formally.

These ideas of approximate reasoning are currently cause for controversial discussions. On the one hand, it is argued that soundness and completeness of Semantic Web reasoning is not to be sacrificed at all, in order to stay within the precise bounds of the specified formal semantics. On the other hand, it is argued that the nature of many emerging Semantic Web applications involves data which is not necessarily entirely accurate, and at the same time is critical in terms of response time, so that sacrificing reasoning precision appears natural [1].

Another suggestion to avoid the necessity is to restrict knowledge representation to so-called tractable fragments that allow for fast sound and complete reasoning. Although this might be useful in scenarios where all essential knowledge can be modelled within the restricted fragment, in general there are strong arguments in favor of the usage of expressive formalisms:

- Real and comprehensive declarative modelling should be possible. A content expert wanting to describe a domain as comprehensive and as precisely as possible will not want to worry about limiting scalability or computability effects.
- As research proceeds, more efficient reasoning algorithms might become available that could be able to more efficiently deal with expressive specification formalisms. Having elaborated specifications at hand enables to reuse the knowledge in a more advanced way.
- Finally, elaborated knowledge specifications using expressive logics can reduce engineering effort by horizontal reuse: Knowledge bases could then be employed for different purposes because the knowledge is already there. However, if only shallow modelling is used, updates would require overhead effort.

From our perspective, it depends on the specifics of the problem at hand whether approximate reasoning solutions can or should be used. We see clear potential in the fields of information retrieval, semantic search, as well as ontology engineering support, to name just a few examples.

At the same time, however, we would like to advocate that allowing for unsound and/or incomplete reasoning procedures in such applications must not lead to arbitrary "guessing" or to deduction algorithms which are not well-understood. Quite on the contrary, we argue that in particular for approximate reasoning, it is of utmost importance to provide ways of determining how feasible the approximations are, i.e. of what quality the answers given by such algorithms can be expected to be.

Obviously, soundness and completeness with respect to the given formal semantics of the underlying knowledge representation languages cannot be used as a measure for assessing the quality of approximate reasoning procedures. Instead, they must be evaluated experimentally, and analysed by statistical means.

In this paper, we thus lay the foundations for a statistical approach to evaluating approximate reasoning algorithms. We will do this in a very abstract manner, which can be made concrete in different ways, depending on the considered use case. At the same time, we will use this statistical perspective to precisely define approximate reasoning notions which to date have remained quite vague. We furthermore show that our mathematical modelling can be used for guiding the development of composed approximate reasoning systems. In the end, our mathematical modelling can be used for rigorous comparative statistical evaluation of approximate reasoning algorithms.

As a word of caution, let us remark that the notion *approximate reasoning* bears two different meanings in two different communities. Often, the notion is associated with *uncertainty reasoning* e.g. in the sense of fuzzy or probabilistic approaches. The notion of approximate reasoning we use in this paper refers to approximate reasoning algorithms on data which is *not* uncertain in this sense.[1]

While approximate reasoning methods for propositional and first-order logic have been proposed (see e.g. [2,3,4,5,6,7,8,9,10]), they are only now being applied in the context of OWL reasoning for Semantic Web technologies. Notable recent papers papers in this area are [11,12,13,14,15,16,17,18] — and to the best of our knowledge, this list should be almost exhaustive.

The paper is structured as follows. In Section 2, we will establish a mathematical framework as a foundation for approximate reasoning notions and evaluation. In Section 3 we will discuss composition of approximate reasoning algorithms from the perspective of our framework. In Section 4 we show how to instantiate our framework by means of an example. We conclude in Section 5.

[1] Perhaps introducing the notion of *qualitative approximate reasoning* – to replace *approximate reasoning* in our sense – would help to clarify matters. In order to be consistent with the literature, however, we prefer to use the established notion for now.

2 A Mathematical Framework for the Study of Approximate Reasoning

In this section, we will establish a mathematical framework. By doing this, we will provide a formal basis for central notions of the field and establish guidance for lines of further research in that area.

First, let us stipulate some abbreviations which we will use in the sequel: let $\mathbb{R}^+ = \{x \in \mathbb{R} : x \geq 0\}$ and $\mathbb{R}^+_\infty = \{x \in \mathbb{R} : x \geq 0\} \cup \{+\infty\}$.

First of all, we have to come up with a general and generic formalization of the notion of a *reasoning task*. Intuitively, this is just a *question* (or query) posed to a system that manages a knowledge base, which is supposed to deliver an *answer* after some processing *time*. The (maybe gradual) validity of the given answer can be evaluated by investigating its compliance with an abstract semantics. We will extend this classical conceptualisation in the following way: we allow an algorithm to – roughly spoken – change or refine its output as time proceeds, thus capturing the notion of *anytime behaviour*, as a central concept in approximate reasoning. Yet in doing so, we have to take care not to lose the possibility of formalizing "classical" termination. We solve this by stipulating that every output of the system shall be accompanied by the information, whether this output is the ultimate one.

In the sequel we will formalize those intuitions. By the term INPUT SPACE we denote the set of possible concrete reasoning tasks. Formally, we define the input space as a probability space (Ω, P), where Ω is some set (of inputs) and P is a probability measure on Ω. The probability $P(\omega)$ encodes how often a specific input (knowledge base, query) ω occurs in practice resp. how relevant it is for practical purposes. Naturally, information about the probability distribution of inputs will be difficult to obtain in practice (since, e.g., in general there can be infinitely many different inputs). So rules of thumb, like giving short queries a higher probability than long ones, or using some kind of established benchmarks, will have to be used until more systematic data is available.

The use of having a probability on the set of inputs is quite obvious: as already stated before, correctness of results cannot be guaranteed in the approximate case. So in order to estimate how good an algorithm performs in practice, it is not only important, how much the given answer to a specific input deviates from the correct one, but also how likely (or: how often) that particular input will be given to the system. Certainly, a wrong (or strongly deviant) answer to an input will be more tolerable if it occurs less often.

For actual evaluations, one will often use a discrete probability space. For the general case – for developing the theory in the sequel – we will assume that all occurring functions are measurable (i.e. integrals over them exist), which is obviously a very mild assumption from a computer science perspective.

The OUTPUT SPACE comprises all possible answers to any of the problems from the input space. In our abstract framework, we define it simply as a set X. A function $e : X \times X \to \mathbb{R}^+$ – which we call *error function* – gives a quantitative measure as to what extent an output deviates from the desired output (as given by a sound and complete algorithm). More precisely, the real number $e(x, y)$

stands for the error in the answer x, assuming that y would be the correct answer. For all $x \in X$ we assume $e(x, x) = 0$, but we place no further constraints on e. It will be determined by the problem under investigation, though a suitable example could be $1 - f$, where f is the *f-measure* as known from information retrieval. In cases, it might be also useful to put more constraints on the error function, one could e.g. require it to be a metric,[2] if the output space has a structure where this seems reasonable.

We will assess the usefulness of an approximate reasoning algorithm mainly by looking at two aspects: Runtime and error when computing an answer. By introducing the error function, we are able to formalize the fact that out of two wrong answers one might still be better than the other since it is "closer" to the correct result. While this might not seem to make much sense in some cases (e.g. when considering the output set $\{true, false\}$ or other nominal scales[3]), it might by quite valuable in others: When we consider an instance retrieval task, the outputs will be sets of domain individuals. Obviously, one would be more satisfied with an answer where just one element out of hundred is missing (compared to the correct answer) than with a set containing, say, only non-instances.

We assume X to contain a distinguished element $\perp$ which denotes *no output*. This is an issue of "backward compatibility", since classical algorithms – and also many approximate reasoning algorithms – usually do not display any output until termination. So, to include them into our framework, we define them to deliver $\perp$ before giving the ultimate result. $\perp$ will also be used as output value in case the algorithm does not terminate on the given input.

Since by this definition, $\perp$ contains no real information, one could argue about additional constraints for the error function with respect to this distinguished element, e.g., $e(\perp, y) \geq \sup_{x \in X}\{e(x, y)\}$ or even $e(\perp, y) \geq \sup_{x,z \in X}\{e(x, z)\}$. We do not need to impose these in general, however.

After having formalized inputs and outputs for problems, we now come to the actual algorithms. In order not to unnecessarily overcomplicate our formal considerations, we make some additional assumptions: We assume that hardware etc. is fixed, i.e., in our abstraction, an algorithm is always considered to include the hard- and software environment it is run in. I.e., we can, for example, assign any algorithm-input pair an exact runtime (which may be infinite). This assumption basically corresponds to a "laboratory" setting for experiments, which abstracts from variables currently not under investigation.

So, let $\mathcal{A}$ be a set of algorithms. To every algorithm $a \in \mathcal{A}$ we assign an IO-FUNCTION $f_a : \Omega \times \mathbb{R}^+ \to X \times \mathbf{2}$ with $\mathbf{2} := \{0, 1\}$. Hereby, $f_a(\omega, t) = (x, b)$ means that the algorithm a applied to the input (task, problem, ...) ω yields the result x after running time t together with the information whether the algorithm has already reached its final output ($b = 1$) or not yet ($b = 0$). As a natural constraint, we require f_a to additionally satisfy the condition that for all $t_2 \geq t_1$ we have that

[2] I.e. A distance function as used in the mathematical theory of metric spaces.

[3] Although also these cases can seamlessly be covered by choosing a discrete error function.

$$f_a(\omega, t_1) = (x, 1) \text{ implies } f_a(\omega, t_2) = (x, 1),$$

i.e. after having indicated termination, the output of the algorithm (including the termination statement) will not change anymore. For convenience we write $f_a^{\mathrm{res}}(\omega, t) = x$ and $f_a^{\mathrm{term}}(\omega, t) = b$, if $f_a(\omega, t) = (x, b)$.

By $f_0 : \Omega \to X$ we denote the CORRECT OUTPUT FUNCTION, which is determined by some external specification or formal semantics of the problem. This enables us to verify the (level of) correctness of an answer $x \in X$ with respect to a particular input ω by determining $e(x, f_0(\omega))$ – the smaller the respective value, the better the answer. By our standing condition on e, $e(x, f_0(\omega)) = 0$ ensures $f_0(\omega) = x$ coinciding with the intuition.

To any algorithm a, we assign a RUNTIME FUNCTION $\varrho_a : \Omega \to \mathbb{R}^+_\infty$ by setting

$$\varrho_a(\omega) = \inf\{t \mid f_a^{\mathrm{term}}(\omega, t) = 1\},$$

being the smallest time, at which the algorithm a applied to input ω indicates its termination.[4] Note that this implies $\varrho_a(\omega) = \infty$ whenever we have $f_a^{\mathrm{term}}(\omega, t) = 0$ for all $t \in \mathbb{R}^+$. Algorithms, for which for all $\omega \in \Omega$ we have that $\varrho_a(\omega) < \infty$ and $f_a^{\mathrm{res}}(\omega, t) = \perp$ for all $t < \varrho_a(\omega)$ are called ONE-ANSWER ALGORITHMS: They give only one output which is not $\perp$, and are guaranteed to terminate[5] in finite time.

Clearly, for a given time t, the expression $e(f_a^{\mathrm{res}}(\omega, t), f_0(\omega))$ provides a measure of how much the current result provided by the algorithm diverges from the correct solution. Moreover, it is quite straightforward to extend this notion to the whole input space (by taking into account the occurrence probability of the single inputs). This is done by the next definition.

The DEFECT $\delta(a, t)$ ASSOCIATED WITH AN ALGORITHM $a \in \mathcal{A}$ AT A TIME POINT t is given by

$$\delta : \mathcal{A} \times \mathbb{R}^+ \to \mathbb{R}^+_\infty : \delta(a, t) = E(e(f_a^{\mathrm{res}}(\omega, t), f_0(\omega))) = \sum_{\omega \in \Omega} e(f_a^{\mathrm{res}}(\omega, t), f_0(\omega)) P(\omega).$$

Note that E denotes the expected value, which is calculated by the rightmost formula.[6] Furthermore, one can even abstract from the time and take the results after waiting "arbitrarily long": The (ULTIMATE) DEFECT of an algorithm $a \in \mathcal{A}$ is given by

$$\delta : \mathcal{A} \to \mathbb{R}^+_\infty : \delta(a) = \limsup_{t \to \infty} \delta(a, t).$$

[4] We make the reasonable assumption that f_a^{res} is right-continuous.

[5] We impose termination here because our main interest is in reasoning with description logics for the Semantic Web. The same notion *without* imposing termination would also be reasonable, for other settings.

[6] The sum could easily be generalised to an integral – with P being a probability measure –, however it is reasonable to expect that Ω is discrete, and hence the sum suffices.

By the constraint put on the IO-function we get

$$\delta(a) = E(e(f_a^{\mathrm{res}}(\omega, \varrho_a(\omega)), f_0(\omega))) = \sum_{\omega \in \Omega} e(f_a^{\mathrm{res}}(\omega, \varrho_a(\omega)), f_0(\omega))P(\omega).$$

if a terminates for every input.

2.1 Comparing Algorithms after Termination

For $a, b \in \mathcal{A}$, we say that a is MORE PRECISE THAN b if it has smaller ultimate defect, i.e. if

$$\delta(a) \leq \delta(b).$$

Furthermore, it is often interesting to have an estimate on the runtime of an algorithm. Again it is reasonable to incorporate the problems' probabilities into this consideration. So we define the AVERAGE RUNTIME[7] of algorithm a by

$$\alpha(a) = E(\varrho_a(\omega)) = \sum_{\omega \in \Omega} \varrho_a(\omega)P(\omega).$$

This justifies to say that a is QUICKER THAN b if

$$\alpha(a) \leq \alpha(b).$$

Note that this does not mean that a terminates earlier than b on every input. Instead, it says that when calling the algorithm very often, the overall time when using a will be smaller than when using b – weighted by the importance of the input as measured by P.

Throughout the considerations made until here, it has become clear that there are two dimensions along which approximate reasoning algorithms can be assessed or compared: runtime behaviour and accuracy of the result. Clearly, an algorithm will be deemed better, if it outperforms another one with respect to the following criterion:

Definition 1. *For $a, b \in \mathcal{A}$, we say that a* IS STRONGLY BETTER THAN *b if a is more precise than b and a is quicker than b.*

The just given definition is very strict; a more flexible one will be given below, when we introduce the notion that an algorithm a is *better than* an algorithm b.

2.2 Anytime Behaviour

The definitions just given in Section 2.1 compare algorithms after termination, i.e. anytime behaviour of the algorithms is not considered. In order to look at anytime aspects, we need to consider the continuum of time points from initiating the anytime algorithm to its termination.

[7] We are aware that in some cases, it might be more informative to estimate the runtime behaviour via other statistical measures as e.g. the median.

For $a, b \in \mathcal{A}$, we say that a is MORE PRECISE THAN b AT TIME POINT t if it has smaller defect wrt. a and t, i.e. if

$$\delta(a,t) \leq \delta(b,t).$$

We say that $a \in \mathcal{A}$ REALISES A DEFECTLESS APPROXIMATION if

$$\lim_{t\to\infty} \delta(a,t) = 0.$$

Note that $\delta(a) = 0$ in this case.

Definition 2. *We say that an algorithm $a \in \mathcal{A}$ is an* ANYTIME ALGORITHM *if it realizes a defectless approximation. We say that it is a* MONOTONIC ANYTIME ALGORITHM *if it is an anytime algorithm and furthermore $\delta(a,t)$ is monotonically decreasing in t, i.e. if $\delta(a,\cdot) \searrow 0$.*

Obviously, is is reasonable to say about two algorithms a and b – be they anytime or not –, that (1) a is better than b if a is more precise than b at any time point. A less strict – and apparently more reasonable – requirement accumulates the difference between a and b over the entire runtime, stating that (2) a is better than b if $\sum_{\omega\in\Omega} P(\omega) \int_{t=0}^{\max\{\varrho_a(\omega),\varrho_b(\omega)\}} (e(f_a^{\text{res}}(\omega,t), f_0(\omega)) - e(f_b^{\text{res}}(\omega,t), f_0(\omega))\mathrm{d}t \leq 0$. We find formula (2) still not satisfactory as it ignores the reasonable assumption that some time points might be more important than others, i.e. they need to be weighted more strongly. Formally, this is done by using a different measure for the integral or – equivalently – a density function $\bar{f} : \mathbb{R}^+ \to \mathbb{R}^+$, which modifies the integral. Summarizing, we now define for two (not necessarily anytime) algorithms a and b that (3) a IS BETTER THAN b (wrt. a given density function $\bar{f}$) if

$$\sum_{\omega\in\Omega} P(\omega) \int_{t=0}^{\max\{\varrho_a(\omega),\varrho_b(\omega)\}} \big(e(f_a^{\text{res}}(\omega,t), f_0(\omega)) - e(f_b^{\text{res}}(\omega,t), f_0(\omega))\big)\bar{f}(t)\mathrm{d}t \leq 0.$$

Our definition (3) specialises to the case in (2) for the constant density function $\bar{f} \equiv 1$. We cannot capture (1) with our definition by one specific choice of $\bar{f}$, so in the case of (1) we simply say that a is more precise than b at any time point.[8]

Clearly, the choice of the density function depends on the considered scenario. In cases where only a fixed time t_{timeout} can be waited before a decision has to be made based on the results acquired so far, the value $\bar{f}(t)$ of density function would be set to zero for all $t \geq t_{\text{timeout}}$. Usually earlier results are preferred to later ones which would justify the choice of an $\bar{f}$ that is monotonically decreasing.

3 Anytime Algorithms by Composition

Realised approximate reasoning systems are often not anytime. However, it is possible to obtain anytime behaviour by composing one-answer algorithms.

[8] However, (1) could be formulated in terms of (3) as a being better than b for all Dirac delta functions that have their singularity at a nonnegative place.

Assume that a number of algorithms a_i $(i = 1, \ldots, n)$ is given. Furthermore, assume there is an ORACLE ALGORITHM **c** whose behaviour can be described by a function $c : (X \times 2)^n \to X \times 2$ which combines a vector of outputs $(a_1(\omega, t), \ldots, a_n(\omega, t))$ of the algorithms a_i and yields a single output. Given an input ω, the invocation of all a_i in parallel and the subsequent call of the oracle algorithm yield a new algorithm $c_{a_1,\ldots,a_n}$ with IO-function

$$f_{c_{a_1,\ldots,a_n}}(\omega, t) = c(a_1(\omega, t), \ldots, a_n(\omega, t)).$$

The definition just given is very general in order to allow for a very free combination, depending on the algorithms which are being combined. For the general setting, we impose only the very general constraint that for all $x_1, \ldots, x_n \in X$ we have

$$c((x_1, 1), \ldots, (x_n, 1)) = (x, 1)$$

for some x, and also that the natural constraint from page 155 on the corresponding IO-function $f_{c_{a_1,\ldots,a_n}}$ is satisfied. This is just to ensure $\varrho_{c_{a_1,\ldots,a_n}}(\omega) \leq \max\{\varrho_{a_1}, \ldots, \varrho_{a_n}\}$, i.e. the "combiner" indicates termination at the latest whenever all of the single input algorithms a_i do so.

It is more interesting to look at more concrete instances of oracles. Assume now that $a_1, \ldots, a_{n-1}$ are one-answer algorithms and that a_n is an (always terminating) sound and complete algorithm. Let **c** be such that

$$c(a_1(\omega, \varrho_{a_n}(\omega)), \ldots, a_{n-1}(\omega, \varrho_{a_n}(\omega)), a_n(\omega, \varrho_{a_n}(\omega))) = (f^{\text{res}}_{a_n}(\omega), 1).$$

Then it is easy to see that $c_{a_1,\ldots,a_n}$ is anytime.

If we know about soundness or completeness properties of the algorithms $a_1, \ldots, a_{n-1}$, then it is also possible to guarantee that $c_{a_1,\ldots,a_n}$ is monotonic anytime. This can be achieved in several ways, and we give one specific example based on ABox reasoning in description logics:

Assume that each input consist of a class description C over some description logic L, and each output consists of a set of (named) individuals. For constructing an oracle from such algorithms, we will actually consider as outputs *pairs* (A, B) of sets of individuals. Intuitively, A contains only individuals which are *known* to belong to the extension of C, while B constitutes an individual set which *is known to contain* all individuals in the extension of C. A single output (set) A can be equated with the output pair (A, A). Now let $a_1, \ldots, a_n$ be sound[9] but incomplete[10] one-answer algorithms over L, let $b_1, \ldots, b_m$ be complete but unsound one-answer algorithms over L and let a be a sound, complete and terminating algorithm over L, i.e. $f^{\text{res}}_a(C, \varrho_a)$ – which we denote by C_a – contains exactly all named individuals that are in the extension of C as a logical

[9] We mean soundness in the following sense: If the set I of individuals is the correct answer, then the algorithms yields as output a pair (A, A) of sets with $A \subseteq I$.

[10] We mean completeness in the following sense: If the set I of individuals is the correct answer, then the algorithms yields as output a pair (A, A) of sets with $I \subseteq A$.

consequence of the given knowledge base. Under this assumption, we know that $f^{\text{res}}_{a_i}(C, \varrho_{a_i}) = (C_{a_i}, \mathbf{I})$ and $f^{\text{res}}_{b_j}(C, \varrho_{b_j}) = (\emptyset, C_{b_j})$ for some sets C_{a_i} and C_{b_j}, where I stands for the set of all (known) individuals, and furthermore we know that $C_{a_i} \subseteq C_a \subseteq C_{b_j}$ for all i, j.

The oracle **c** is now defined as follows.

$$\mathbf{c}(a_1(C,t),\ldots,a_n(C,t),b_1(C,t),\ldots,b_m(C,t),a(C,t))$$
$$= \begin{cases} ((f^{\text{res}}_a(C,t), f^{\text{res}}_a(C,t)), 1) & \text{for } t \geq \varrho_a(C), \\ ((\mathit{upper}, \mathit{lower}), \mathit{term}) & \text{for } t < \varrho_a(C) \\ \qquad \text{where } \mathit{lower} = \bigcup_{(A_i,B_i,1)=f_{a_i}(C,t)} A_i, & \\ \qquad\qquad \mathit{upper} = \bigcap_{(A_j,B_j,1)=f_{b_j}(C,t)} B_j, & \\ \qquad\qquad \mathit{term} = 1 \text{ if } \mathit{lower} = \mathit{upper}, \text{ otherwise } 0. & \end{cases}$$

Note that the empty set union is by definition the empty set, while the empty set intersection is by definition I.

In words, the oracle realises the following behaviour: if the sound and complete subalgorithm has terminated, display its result. Before, use the lower resp. upper bounds delivered by the sound resp. complete algorithms to calculate one intermediate lower and one intermediate upper bound. If those two happen to coincide, the correct result has been found and may terminate without waiting for a's termination. This *squeezing in* of the correct result now also explains why we have chosen to work with pairs of sets as outputs.

As error function, we might use the sum of the symmetric difference between A and A_0, respectively between B and A_0, i.e.

$$e((A,B),(A_0,A_0)) = |A_0 \setminus A| + |B \setminus A_0|.$$

We could also use a value constructed from similar intuitions like precision and recall in information retrieval, but for our simple example, this error function suffices. It is indeed now straightforward to see that $\mathbf{c}_{a_1,\ldots,a_n,b_1,\ldots,b_m,a}$ is monotonic anytime. It is also clear that $\mathbf{c}_{a_1,\ldots,a_n,b_1,\ldots,b_m,a}$ is more precise than any of the a_i and b_j, at all time points.

4 An Example

In this section, we will instantiate the very general framework established in the preceding sections. We will use the presented techniques to compare three approximate reasoning algorithms and compose a (simple) anytime algorithm following the example at the end of Section 3.

Consider the three algorithms Screech-all, Screech-none and KAON2, as discussed in [19]. We do not intend to give any details here, and it shall suffice to mention that these are one-answer algorithms for reasoning with the description logic $\mathcal{SHIQ}$, and the task considered is instance retrieval for named classes. Screech-all is complete but unsound, Screech-none is sound but incomplete, and KAON2 is sound and complete.

Following the general framework, we first have to stipulate the probability space (Ω, P) for our case. Here we introduce the first simplifying assumptions, which are admittedly arguable, but will suffice for the example:

– We consider only one knowledge base, namely the well-known Wine ontology. Further evaluation data is available [19] but will not be taken into account for the illustrating example.
– As queries, we consider only instance retrieval tasks, i.e. given an atomic class description, we query for the set of individuals which can be inferred to be instances of that class. Hence Ω – the query space – consists of named classes $\mathbf{C}$ of the Wine ontology the instances of which are to be retrieved: $\Omega = \mathbf{C}$. Examples for named classes in this ontology are e.g. `Chardonnay`, `StEmilion` or `Grape`.
– All those instance retrieval queries $\omega \in \Omega$ are assumed to be equally probable to be asked to the system, hence

$$P(\omega) = \frac{1}{|\mathbf{C}|} \text{ for all } \omega \in \Omega.$$

Obviously, the probability of a query could also be assumed differently, e.g. correlating with the number of instances the respective class has. Nevertheless, for the sake of simplicity we will stick to the equidistributional approach.

Obviously, the output space X consists of subsets of the set of individuals $\mathbf{I}$ from the Wine ontology together with the no-output symbol $\perp$: $X = 2^{\mathbf{I}} \cup \{\perp\}$. As the error function e comparing an algorithm's output I with the correct one I_0, we use the inverted value of the common f-measure, i.e.

$$e(I, I_0) := 1 - \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}}$$

where (as usual)

$$\mathit{precision} := \frac{|I \cap I_0|}{|I|} \qquad \text{and} \qquad \mathit{recall} := \frac{|I \cap I_0|}{|I_0|}.$$

According to the proposed handling of $\perp$, we stipulate the overall "worst-case distance": $e(\perp, I_0) = 1$ for all $I \subseteq \mathbf{I}$.

As mentioned before, the set $\mathcal{A}$ of considered algorithms comprises three items:

$$\mathcal{A} = \{\text{KAON2}, \textsc{Screech-all}, \textsc{Screech-none}\}$$

For every of those algorithms we carried out comprehensive evaluations: we queried for the class extensions of every named class and stored the results as well as the time needed. By their nature none of the considered algorithms exhibits a genuine anytime behavior, however, instead of displaying the "honest" $\perp$ during their calculation period, they could be made to display an arbitrary intermediate result. It is straightforward to choose the empty set in order to obtain better results: most class extensions will be by far smaller than half of the individual set, hence the distance of the correct result to the empty set will be a rather good guess.

Hence, for any algorithm a of the above three and any class name C let I_C denote be the set of retrieved instances and t_C denote the measured runtime for accomplishing this task. Then we can define the IO-function as

$$f_a(C,t) = \begin{cases} (\emptyset, 0) \text{ if } t < t_C \\ (I_C, 1) \text{ otherwise.} \end{cases}$$

The values of the correct output function f_0 can be found via KAON2, as this algorithm is known to be sound and complete. Moreover, the runtime functions $\rho_a(C)$ of course coincide in our case with the runtimes t_C measured in the first place. Since all of the considered algorithms are known to terminate, no ρ_a will ever take the value ∞.

After this preconsiderations, we are ready to carry out some calculations estimating the quality of the considered algorithms. Figure 1 shows a plot depicting the decrease of the defect for all the three algorithms. As expected, there is an ultimate defect for the two screech variants, namely 0.013 for SCREECH-NONE and 0.015 for SCREECH-ALL, i.e. with respect to the terminology introduced earlier, we can say that SCREECH-NONE is more precise than SCREECH-ALL. While the defect of KAON2 is initially greater than those of the screech variants, it becomes better than them at about 6 seconds and decreases to zero defect after about 7 seconds. In other words, SCREECH-ALL is more precise than KAON2

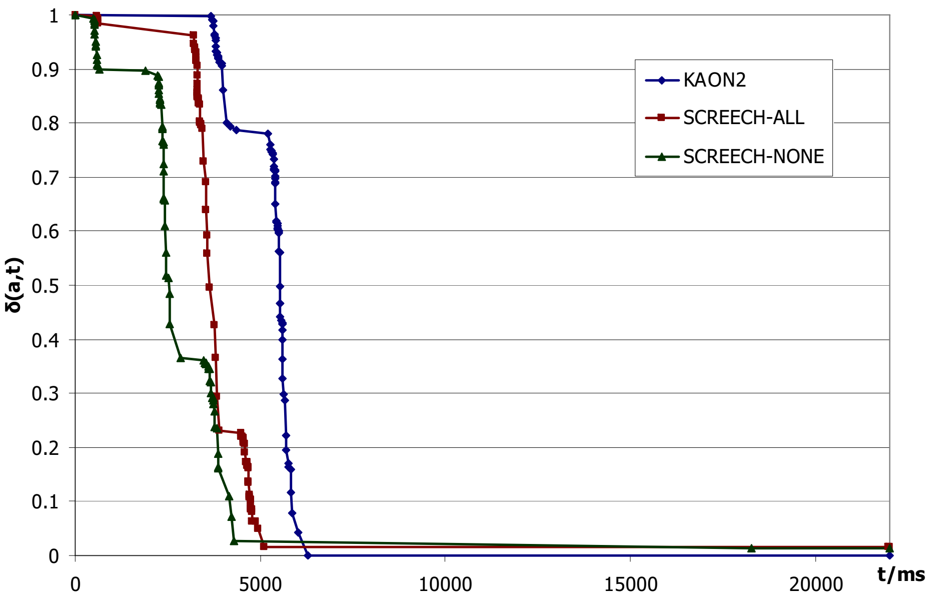


Fig. 1. Defect over time

at all time points less than 6 seconds. A first conclusion from this would be: if a user is willing to wait for 7 seconds for an answer (which then would be guaranteed to be correct) KAON2 would be the best choice, otherwise (if time is crucial and precision not), SCREECH-ALL might be a better choice as it shows the quickest defect decrease.

If we now assume a time-critical application where responses coming in later than, say, 5 seconds are ignored, we can describe this by the fact that SCREECH-ALL is better than KAON2 with respect to the density function

$$\bar{f}(x) = \begin{cases} 1 & 0 \leq x \leq 5, \\ 0 & \text{otherwise.} \end{cases}$$

Considering the fact that SCREECH-ALL is complete, SCREECH-NONE is sound, and KAON2 is both, we can now utilise a variant of the oracle given in the example from Section 3. The behaviour of the combined algorithm can in this simple case be described as follows. It indicates termination whenever one of the following occurs:

– KAON2 has terminated. Then the KAON2 result is displayed as solution.
– Both SCREECH-ALL and SCREECH-NONE have terminated with the same result. In this case, the common result will be displayed as the final one.

If none of above is the case, the experimental findings suggest to choose the SCREECH-NONE result as intermediate figure. The algorithm obtained that way is anytime and more (or equally) precise than any of the single algorithms at all time points.

5 Conclusions

Approaches to approximate reasoning tackle the problem of scalability of deducing implicit knowledge. Especially if this is done on the basis of large-scale knowledge bases or even the whole Web, often the restriction to 100% correctness has to be abandoned for complexity reasons, in particular if quick answers to posed questions are required. Anytime algorithms try to fulfill both needs (speed and correctness) by providing intermediate results during runtime and continually refining them.

In our paper, we have provided solid mathematical foundations for the assessment and comparison of approximate reasoning algorithms with respect to correctness, runtime and anytime behaviour. We are confident that this general framework can serve as a means to classify algorithms w.r.t. their respective characteristics and help in deciding which algorithm best matches the demands of a concrete reasoning scenario.

As opposed to our example scenario, in most practical cases, it will be unfeasible or even impossible to measure the whole input space as it will be too large or even infinite. That is where statistical considerations come into play: one has to identify and measure representative samples of the input space. The first part of this is far from trivial: for fixed settings with frequently queried knowledge bases, such a sample could be determined by protocolling the actually posed queries over a certain period of time. Another way would be to very roughly estimate a distribution based on plausible arguments. Respective heuristics would be: (1) the more complex a query the more unlikely, (2) queries of similar structure are similarly frequent resp. likely, (3) due to some bias in human conceptual thinking, certain logical connectives (e.g. conjunction) are preferred to others (e.g. disjunction, negation) which also gives an opportunity to estimate a query's frequency based on the connectives it contains. Admittedly, those heuristics are still rather vague and more thorough research is needed to improve reliability of such estimates.

In general, the proposed intelligent combination of several algorithms with different soundness/completeness properties (as well as being specialised to certain logical fragments) can increase speed and might help avoid heavy-weight reasoning in cases. We are confident, that this idea can be easily generalised to reasoning tasks other than instance retrieval. Obviously, this strategy comes with an immediate opportunity of parallelisation even if the single algorithms have to be treated as black boxes. Hence, this approach could also be conceived as a somewhat exotic approach to distributed reasoning.

References

1. Fensel, D., Harmelen, F.V.: Unifying reasoning and search to web scale. IEEE Internet Computing 11, 94–96 (2007)
2. Dowling, W.P., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. Journal of Logic Programming 1, 267–284 (1984)
3. Horvitz, E.J.: Reasoning about beliefs and actions under computational resource constraints. In: Kanal, L.N., Levitt, T.S., Lemmer, J.F. (eds.) Uncertainty in Artificial Intelligence 3, pp. 301–324. Elsevier, Amsterdam (1987)
4. Selman, B., Kautz, H.A.: Knowledge compilation using Horn approximations. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991), pp. 904–909 (1991)
5. Schaerf, M., Cadoli, M.: Tractable reasoning via approximation. Artificial Intelligence 74, 249–310 (1995)
6. Cadoli, M., Schaerf, M.: Approximate inference in default reasoning and circumscription. Fundamenta Informaticae 23, 123–143 (1995)
7. Dalal, M.: Anytime clausal reasoning. Annals of Mathematics and Artificial Intelligence 22(3–4), 297–318 (1998)
8. Cadoli, M., Scarcello, F.: Semantical and computational aspects of Horn approximations. Artificial Intelligence 119 (2000)
9. van Harmelen, F., ten Teije, A.: Describing problem solving methods using anytime performance profiles. In: Proceedings of ECAI 2000, Berlin, August 2000, pp. 181–186 (2000)
10. Groot, P., ten Teije, A., van Harmelen, F.: Towards a structured analysis of approximate problem solving: a case study in classification. In: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), Whistler, Colorado (2004)

11. Stuckenschmidt, H., van Harmelen, F.: Approximating terminological queries. In: Larsen, H., et al. (eds.) FQAS 2002. LNCS (LNAI), vol. 2522. Springer, Heidelberg (2002)
12. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL reasoning with large numbers of individuals. In: Proceedings of the International Workshop on Description Logics, DL 2004, Whistler, Canada, pp. 31–40 (2004)
13. Groot, P., Stuckenschmidt, H., Wache, H.: Approximating description logic classification for semantic web reasoning. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 318–332. Springer, Heidelberg (2005)
14. Groot, P., Stuckenschmidt, H., Wache, H.: Approximating description logic classification for semantic web reasoning. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532. Springer, Heidelberg (2005)
15. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 383–397. Springer, Heidelberg (2005)
16. Pan, J.Z., Thomas, E.: Approximating OWL-DL ontologies. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 22-26, 2007, pp. 1434–1439. AAAI Press, Menlo Park (2007)
17. Wache, H., Groot, P., Stuckenschmidt, H.: Scalable instance retrieval for the semantic web by approximation. In: Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z. (eds.) WISE 2005 Workshops. LNCS, vol. 3807, pp. 245–254. Springer, Heidelberg (2005)
18. Stuckenschmidt, H.: Partial matchmaking using approximate subsumption. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 22-26, 2007, pp. 1459–1464. AAAI Press, Menlo Park (2007)
19. Tserendorj, T., Rudolph, S., Krötzsch, M., Hitzler, P.: Approximate OWL reasoning with Screech. In: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems, RR 2008, October 2008, Karlsruhe, Germany (to appear, 2008)

Approximate OWL-Reasoning with Screech[⋆]

Tuvshintur Tserendorj[1], Sebastian Rudolph[2], Markus Krötzsch[2], and Pascal Hitzler[2]

[1] FZI Karlsruhe, Germany
[2] AIFB, University of Karlsruhe, Germany

Abstract. With the increasing interest in expressive ontologies for the Semantic Web, it is critical to develop scalable and efficient ontology reasoning techniques that can properly cope with very high data volumes. For certain application domains, approximate reasoning solutions, which trade soundness or completeness for inctreased reasoning speed, will help to deal with the high computational complexities which state of the art ontology reasoning tools have to face. In this paper, we present a comprehensive overview of the Screech approach to approximate reasoning with OWL ontologies, which is based on the KAON2 algorithms, facilitating a compilation of OWL DL TBoxes into Datalog, which is tractable in terms of data complexity. We present three different instantiations of the Screech approach, and report on experiments which show that the gain in efficiency outweighs the number of introduced mistakes in the reasoning process.

1 Introduction

Scalability of reasoning remains one of the major obstacles in leveraging the full power of the Web Ontology Language OWL [1] for practical applications. Indeed, large-scale applications normally use only a fragment of OWL which is very shallow in logical terms, and thus cannot employ the more sophisticated reasoning mechanisms for accessing knowledge which is implicit in knowledge bases. While the use of such shallow techniques already has added value, it would be preferable if the more complex logical constructors in the language could also be used. Consequently, scalability of OWL reasoning needs to be investigated on a broad front in order to advance the state of the art by several orders of magnitude.

Among the many possible approaches to address scalability, one of them concerns the use of logic programming for this purpose. This can be traced back to the work on Description Logic Programs (DLP) [2,3], which are a naive Horn fragment of OWL DL.[1] Along the same lines lies the OWL DL-fragment

⋆ Research reported in this paper was supported by the EU in the IST project NeOn (IST-2006-027595, http://www.neon-project.org/)), by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project, and by the the German Federal Ministry of Education and Research (BMBF) under the Theseus project, http://theseus-programm.de.

[1] Some recent developments can be found in [4,5,6].

Horn-$\mathcal{SHIQ}$ [7,8], which is based on the sophisticated transformation algorithms implemented in the KAON2-system[2] [8,9]. Horn-$\mathcal{SHIQ}$ is strictly more expressive than DLP and allows, for example, the free use of existential role restrictions.

At the same time, a different effort to leveraging Horn logic for OWL reasoning rests on the idea of approximate reasoning, which presupposes an application scenario where speed is so important that it becomes reasonable to allow some incorrect inferences in order to speed up the reasoning. Our implementation is called Screech [10], and it is based on the idea of approximating an OWL DL knowledge base by Horn clauses. Initial experiments reported in [10] – and briefly in [11] – have shown that Screech indeed improves runtime in some cases, but further evaluations had been missing so far.

In this paper, we introduce two new variants of the Screech approach (in Sections 2 and 3), resulting in three related algorithms, which can be used in combination for approximate OWL reasoning. We will then report on experiments (in Section 4) which we performed for all approaches. They show that all three variants of Screech indeed result in significant speed-up under only a very small number of introduced mistakes.

2 The Screech Approach

2.1 The KAON2-Transformation

Reasoning with KAON2 is based on special-purpose algorithms which have been designed for dealing with large ABoxes. They are detailed in [8] and we present a birds' eyes perspective here, which suffices for our purposes. The underlying rationale of the algorithms is that algorithms for deductive databases have proven to be efficient in dealing with large numbers of facts. The KAON2 approach utilises this by transforming OWL DL ontologies to disjunctive datalog, and by the subsequent application of the mentioned and established algorithms for dealing with disjunctive datalog.

A birds' eyes perspective on the KAON2 approach is depicted in Figure 1. KAON2 can handle $\mathcal{SHIQ}(\mathbf{D})$ ontologies, which corresponds roughly to OWL DL without nominals. The TBox, together with a query are processed by the sophisticated KAON2-transformation algorithm which returns a disjunctive datalog program. This, together with an ABox, is then fed into a disjunctive datalog reasoner which eventually returns an answer to the query.

In some cases, e.g. when querying for instances of named classes, the query does not need to be fed into the transformation algorithm but instead needs to be taken into account only by the datalog reaoner. This allows to compute the disjunctive datalog program offline, such that only the disjunctive datalog engine needs to be invoked for answering the query. All experiments we report on have been performed this way, i.e. they assume an offline transformation of the TBox prior to the experiments.

[2] http://kaon2.semanticweb.org

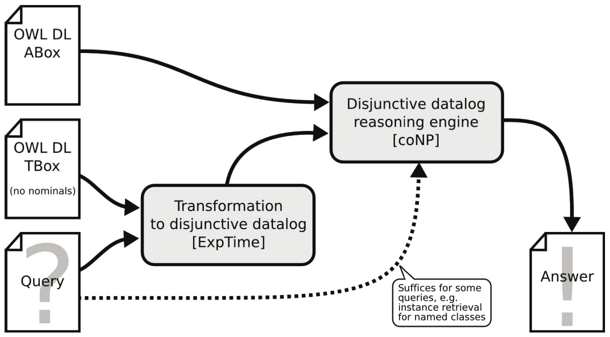

OWL DL ABox
OWL DL TBox (no nominals)
Query
Transformation to disjunctive datalog [ExpTime]
Disjunctive datalog reasoning engine [coNP]
Suffices for some queries, e.g. instance retrieval for named classes
Answer

Fig. 1. KAON2 approach to reasoning

The program returned by the transformation algorithm is in general not logically equivalent to the input TBox. The exact relationship is given below in Theorem 1 due to [8]. Note that statement (b) suffices for our purposes. It also shows that the KAON2 datalog reasoning engine can in principle be replaced by other (sound and complete) reasoning engines without changing the results of the inference process.

Theorem 1. *Let K be a $\mathcal{SHIQ}(\mathbf{D})$ TBox and $D(K)$ be the datalog output of the KAON2 transformation algorithm on input K. Then the following claims hold.*

(a) K is unsatisfiable if and only if $D(K)$ is unsatisfiable.
(b) $K \models \alpha$ if and only if $D(K) \models \alpha$, where α is of the form $A(a)$ or $R(a,b)$, for A a named concept and R a simple role.
(c) $K \models C(a)$ for a nonatomic concept C if and only if, for Q a new atomic concept, $D(K \cup \{C \sqsubseteq Q\}) \models Q(a)$.

Convenient access to the KAON2 transformation algorithm is given by means of the KAON2 OWL Tool[3] dlpconvert,[4] which can also produce F-Logic [12] serialisations which can be used with F-Logic engines like OntoBroker.

2.2 Approximate OWL-Reasoning with SCREECH

Due to the inherent high complexity of reasoning with ontologies, it is to be expected that some application settings will defy even the smartest approaches for achieving sound and complete scalable algorithms. The method of choice for dealing with such situations is to use approximate reasoning, which trades correctness for time, but in a controlled and well-understood way [13]. Approximate Reasoning is indeed recently receiving rising attention from Semantic Web researchers, due to the obvious suitable use cases in this application domain. For some recent work, see e.g. [14,15,16,17,18,19].

[3] http://owltools.ontoware.org/
[4] http://logic.aifb.uni-karlsruhe.de/wiki/Dlpconvert

The SCREECH approach for instance retrieval is based on the fact that data complexity is polynomial for non-disjunctive datalog, while for OWL DL it is coNP complete even in the absence of nominals [7]. SCREECH utilises the KAON2 algorithms, but rather than doing (expensive) exact reasoning over the resulting disjunctive datalog knowledge base, it does approximate reasoning by treating disjunctive rules as if they were non-disjunctive ones, i.e. the disjunctive rules are approximated by Horn rules.

We will first describe the basic variant of SCREECH, which was introduced in [10], and which we call SCREECH-ALL here. SCREECH-ALL is complete, but may be unsound in cases. Its data complexity is polynomial. Two other variants of SCREECH, SCHREECH-NONE and SCREECH-ONE, will be described in Section 3.

SCREECH-ALL uses a modified notion of *split program* [20] in order to deal with the disjunctive datalog. Given a rule

$$H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k,$$

as an output of the KAON2 transformation algorithm, the *derived split rules* are defined as:

$$H_1 \leftarrow A_1, \ldots, A_k \qquad \ldots \qquad H_m \leftarrow A_1, \ldots, A_k.$$

For a given disjunctive program P its *split program* P' is defined as the collection of all split rules derived from rules in P. It can be easily shown that for instance retrieval tasks, the result obtained by using the split program instead of the original one is complete but may be unsound. As the following proposition shows, this is even the case if all integrity constraints, i.e. rules of the form

$$\leftarrow B_1, \ldots, B_n$$

are removed from the split program.

Proposition 1. *Consider a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base K that is logically consistent, let $\mathsf{D}(K)$ denote a disjunctive datalog program obtained by applying KAON2 to K, and let P be the logic program obtained from $\mathsf{D}(K)$ by* SCREECH-ALL. *Then P has a least Herbrand model which satisfies any atomic formula that is true in some minimal Herbrand model of $\mathsf{D}(K)$.*

Especially, P entails all atomic formulae that are true in all (minimal) models of $\mathsf{D}(K)$, i.e. SCREECH-ALL *is complete for instance retrieval on consistent $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases.*

Proof. First, note that we can restrict to propositional programs obtained as the (finite) ground instantiations of the relevant datalog programs. Hence it suffices to consider propositional models.

The fact that P has a least Herbrand model is a standard conclusion from the fact that P is a definite logic program. To show the rest of the claim, consider any minimal Herbrand model $\mathcal{M}$ of the ground instantiation of $\mathsf{D}(K)$ (note that K has some Herbrand model by consistency, and that some of those must be minial since only finitely many ground interpretations exist). Define a ground program $Q_{\mathcal{M}}$ as follows:

$$Q_{\mathcal{M}} = \{H_i \leftarrow B_1 \wedge \ldots \wedge B_m \in P \mid \mathcal{M} \models B_1 \wedge \ldots \wedge B_m \text{ and } \mathcal{M} \models H_i, 1 \leq i \leq n)\}.$$

We claim that $Q_{\mathcal{M}}$ is a definite program with least Herbrand model $\mathcal{M}$. Clearly $Q_{\mathcal{M}}$ is definite (thus has some least Herbrand model), and has $\mathcal{M}$ as a Herbrand model. But obviously any Herbrand model of $Q_{\mathcal{M}}$ that is strictly smaller than $\mathcal{M}$ would also satisfy all rules of $\mathsf{D}(K)$, thus contradicting the assumed minimality of $\mathcal{M}$.

Now clearly $Q_{\mathcal{M}}$ is a subset of the screeched program P, and hence any Herbrand model of P must be greater or equal to the least Herbrand model $\mathcal{M}$ of $Q_{\mathcal{M}}$. Since $\mathcal{M}$ was arbitrary, this shows the claim. □

It is possible to also deal with nominals, i.e. with OWL DL (aka $\mathcal{SHOIN}(\mathbf{D})$) approximately. This was mentioned in [10], but for our purposes it will suffice to consider $\mathcal{SHIQ}$ knowledge bases only, which covers a significant portion of OWL DL.

Putting the pieces together, Screech-all utilises the following subsequent steps for approximate ABox reasoning for $\mathcal{SHIQ}$.

1. Apply transformations as in the KAON2 system in order to obtain a negation-free disjunctive datalog program.
2. Obtain the split program as described above.
3. Do reasoning with the split program, e.g. using the KAON2 datalog engine.

Given a TBox K, the split program obtained from K by steps 1 and 2 is called the *screeched version* of K. The first two steps can be considered to be preprocessing steps for setting up the intensional part of the database. ABox reasoning is then done in step 3. The resulting approach has the following theoretical features:

– It is complete with respect to OWL DL semantics.
– Data complexity is polynomial.

A prototype implementation of our approach is available as the Screech-all OWL approximate reasoner.[5] It is part of the KAON2 OWL Tools. We can convert a $\mathcal{SHIQ}$ ontology into a disjunctive datalog program, e.g. by using the KAON2 OWL Tool `dlpconvert` with the `-x` switch. Screech-all then accesses the results of the translation through the KAON2 API, creates the corresponding split programs and serializes them as Horn logic programs in Edinburgh Prolog syntax or in F-Logic [21,22] syntax. We need to mention, however, that in general support for concrete domains and other features like integrity constraints is not necessarily implemented in off-the-shelf logic programming systems. In these cases, concrete domains etc. cannot be used. The KAON2 OWL Tool `ded`, for example, performs a language weakening step by removing all concrete domains, and may come in handy in such situations.

[5] http://logic.aifb.uni-karlsruhe.de/screech

$$\begin{aligned}
\text{serbian} \sqcup \text{croatian} &\sqsubseteq \text{european} \\
\text{eucitizen} &\sqsubseteq \text{european} \\
\text{german} \sqcup \text{french} \sqcup \text{beneluxian} &\sqsubseteq \text{eucitizen} \\
\text{beneluxian} &\equiv \text{luxembourgian} \sqcup \text{dutch} \sqcup \text{belgian} \qquad (1)
\end{aligned}$$

serbian(ljiljana)	serbian(nenad)	german(philipp)	french(julien)
chinese(yue)	german(peter)	german(stephan)	mongolian(tuvshintur)
indian(anupriya)	belgian(saartje)	german(raphael)	chinese(guilin)

Fig. 2. Example ontology

2.3 A Simple Example

We demonstrate the approach on a simple OWL DL ontology. It contains only a class hierarchy and an ABox, and no roles, but this will suffice to display the main issues.

The ontology is shown in Figure 2, and its intended meaning is self-explanatory. Note that line (1) translates into the four clauses

$$\begin{aligned}
\text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian}(x) &\leftarrow \text{beneluxian}(x), \qquad (2)\\
\text{beneluxian}(x) &\leftarrow \text{luxembourgian}(x),\\
\text{beneluxian}(x) &\leftarrow \text{dutch}(x),\\
\text{and} \qquad \text{beneluxian}(x) &\leftarrow \text{belgian}(x).
\end{aligned}$$

Thus, our approach changes the ontology by treating the disjunctions in line (2) as conjunctions. Effectively, this means that the rule (2) is replaced by the three rules

$$\begin{aligned}
\text{luxembourgian}(x) &\leftarrow \text{beneluxian}(x),\\
\text{dutch}(x) &\leftarrow \text{beneluxian}(x),\\
\text{and} \qquad \text{belgian}(x) &\leftarrow \text{beneluxian}(x).
\end{aligned}$$

This change affects the soundness of the reasoning procedure. However, in the example most of the ABox consequences which can be derived by the approximation are still correct. Indeed, there are only two derivable facts which do not follow from the knowledge base by classical reasoning, namely dutch(saartje) and luxemburgian(saartje). All other derivable facts are correct.

3 Variants of Screech

We will now introduce two other variants of SCREECH, besides SCREECH-ALL introduced above. These other variants are called SCREECH-NONE and SCREECH-ONE.

SCREECH-NONE is defined by simply removing all disjunctive rules (and all integrity constraints) after the transformation by the KAON2-algorithm. For the example from Section 2.3, this means that rule (2) is simply deleted. The resulting reasoning procedure is sound, but incomplete, on $\mathcal{SHIQ}$ knowledge bases.

Screech-one is defined by replacing each disjuntive rules by *exactly one* of the split rules. This selection can be done randomly, but will be most useful if the system has some knowledge – probably of statistical nature – on the size of the extensions of the named classes.[6] For our example from Section 2.3, when considering rule (2) we can use the additional knowledge that there are more dutch people than belgians or luxenbourgians, thus this rule is replaced by the single rule

$$\text{dutch}(x) \leftarrow \text{beneluxian}(x).$$

We also remove all integrity constraints after the translation. The resulting reasoning proceedure is neither sound nor complete. We thus obtain the following result.

Proposition 2. *Instance retrieval with* Schreech-none *is sound but incomplete. Instance retrieval with* Schreech-one *in general is neither sound nor complete.*

Proof. Soundness of Schreech-none is immediate from the fact that calculations are performed on a subset of the computed clauses, together with monotonicity of the employed datalog variant. For all other claims it is easy to find counterexamples. □

The properties of Screech are summarised in Table 1.

Table 1. Screech variants and their basic properties

variant	description	sound	complete
Screech-all	use *all* of the split rules	no	yes
Screech-none	use *none* of the split rules	yes	no
Screech-one	use *one* of the split rules	no	no

From a theoretical point of view, it would be satisfying to characterize the described approximations in terms of extremal bounds in certain logical fragments. However, we remark that the unsound screech variants do not yield greatest Horn lower bounds in the sense of [23] w.r.t. the disjunctive datalog program, not even if we modify the definition to allow only definite Horn rules. As a counterexample for Screech-all, consider the program $\{\leftarrow C(a), C(a) \vee C(b) \leftarrow\}$. Its screeched version is $\{C(a) \leftarrow, C(b) \leftarrow\}$, but its greatest lower bound in the sense of [23] would be $\{C(b) \leftarrow\}$. Analogously, we note that Screech-one yields no greatest lower bound, even if integrity constraints are included (which obviously makes the procedure complete while still being unsound). To see this, consider the program $\{C(a) \leftarrow, C(b) \leftarrow, \leftarrow A(a), \leftarrow B(b), A(x) \vee B(x) \leftarrow C(x)\}$. Its (extended) Screech-one versions are $\{C(a) \leftarrow, C(b) \leftarrow, \leftarrow A(a), \leftarrow B(b), A(x) \leftarrow C(x)\}$ and $\{C(a) \leftarrow, C(b) \leftarrow, \leftarrow A(a), \leftarrow B(b), B(x) \leftarrow C(x)\}$, but its greatest lower bound would be $\{C(a) \leftarrow, C(b) \leftarrow, B(a) \leftarrow, A(b) \leftarrow\}$.

[6] This was suggested by Michael Sintek.

3.1 Expected Results

Prior to performing our experiments – which we will report in Section 4 – we formulated the expected outcome from the different variants of Screech.

- Screech-one – assuming the mentioned knowledge about the size of the extensions of atomic classes – compared to Screech-all should show overall *less* errors for some suitable knowledge bases. We also expected Screech-one to be quicker than Screech-all.
- Screech-none should be quicker than Screech-all and Screech-one. We expected that the number of errors should be comparable with Screech-all, but more severe than Screech-one.

We furthermore expected, that the parallel execution of the two variants Screech-all and Screech-none should help to determine *exact* answers in some cases quicker than using the KAON2 datalog reasoner. This expectation is based on the following fact: If the extensions of some class C as computed by Screech-all and Screech-none are *of the same size*, then the computed extensions are actually correct (sound and complete) with respect to the original knowledge base.

4 Experimental Evaluation

An approximate reasoning procedure needs to be evaluated on real data from practical applications. Handcrafted examples are of only limited use as the applicability of approximate methods depends on the structure inherent in the experimental data.

So we evaluated some popular publicly available ontologies. In some cases we had to cautiously modify them in order to enable KAON2 to perform reasoning tasks on them, but the general approach was to first use KAON2 for transforming the TBoxes to disjunctive datalog. Also offline, a screeched version of the TBox was produced. We then invoked the KAON2 disjunctive datalog engine on both the resulting disjunctive datalog program and on the screeched version, to obtain a comparison of performance.[7]

For all our experiments, we used a T60p IBM Thinkpad with 1.9GB of RAM, with the Java 2 Runtime Environment, Standard Edition (build 1.5.0_09-b03).

4.1 Results in a Nutshell

We performed comprehensive experiments with GALEN, WINE, DOLCE, and SEMINTEC. Before we report in more detail, we list a summary of the results.

- Screech-all shows an average speedup in the range between 8 and 67%, depending on the queried class and the ontology under consideration, while 38 to 100% of the computed answers are correct. Most interestingly, a higher speedup usually seemed to correlate with less errors.

[7] The raw data of the experiments can be found online under `http://logic.aifb.uni-karlsruhe.de/wiki/Screech`.

– Screech-one compared to Screech-all has overall *less* errors. In most cases, all correct class members are retrieved. Runtime is similar to Screech-all.
– Screech-none compared to Screech-all shows similar run-time. In most cases, the extensions are computed *correctly* – with the exception of WINE, for which we get 2% missing answers.
– Running Screech-all and Screech-none in parallel and comparing the results, allows the following: If the computed extensions are of the same size, then we know that all (and only correct) class members have been found. This is the case for more than 76% of all classes we computed.

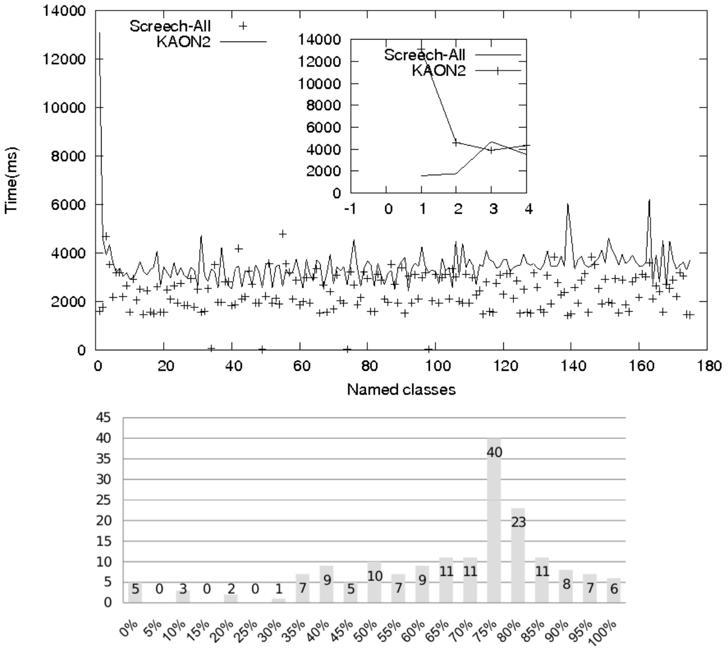


Fig. 3. Performance comparison for SCREECH-ALL and KAON2. Top: absolute retrieval times. Bottom: Screech-ALL retrieval times as percentage of KAON2 retrieval times. The ordinate gives the number of classes for which this percentage was achieved, e.g. for 40 classes the percentage was around 75%.

4.2 GALEN

We first report on our experiments with the OWL DL version of the GALEN Upper Ontology.[8] As it is a TBox ontology only, we populated GALEN's 175 classes randomly with 500 individuals.[9] GALEN does not contain nominals or concrete domains. GALEN has 673 axioms (the population added another 500).

[8] http://www.cs.man.ac.uk/∼rector/ontologies/simple-top-bio/
[9] Using the `pop` KAON2 OWL tool.

After the TBox translation to disjunctive datalog we obtained ca. 1833 disjunctive datalog rules,[10] ca. 52 of which contained disjunctions.[11] The SCREECH-ALL split resulted in 113 new rules, replacing the disjunctive ones. 149 integrity constraints were also removed.

Figure 3 shows the runtime for each named classe taken by SCREECH-ALL and KAON2. Note the times for the first retrieved named class in Figure 3 (top), which is considerably higher for KAON2 than for SCREECH-ALL. The reason for this is that on the first run KAON2 performs the TBox translation (see Figure 1). The translated TBox is then stored, and thus does not need to be repeated for subsequent queries of extensions of named classes according to Theorem 1.

Table 2. Summary of the three SCREECH versions on GALEN. *miss* indicates the elements of the extensions which were *not* found by the approximation, *corr* indicates those which were correctly found, and *more* indicates those which were incorrectly computed to be part of the extension. *time* gives the runtime (in ms) for the respective SCREECH version, while *KAON2* gives the runtime (in ms) using the disjunctive rules. *f-meas* is the f-measure known from information retrieval, computed as (2 · precision · recall)/(precision + recall) with precision = corr/(corr + more) and recall = corr/number of actual instances, *corr.class* gives the fraction of classes for which the extension was computed correctly, and *time/KAON2* is the ratio between time and KAON2.

Variant	miss	corr	more	time	KAON2	f-meas	corr.class	time/KAON2
SCREECH-ALL	0	5187	465	255132	1007527	0.957	0.78	0.25
SCREECH-ONE	5	5182	134	277009	1007527	0.987	0.98	0.27
SCREECH-NONE	10	5177	0	244994	1007527	0.999	0.78	0.24

We then queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits. The relative runtimes by SCREECH-ALL in percentage of KAON2 runtime is displayed in Figure 3 (right). It shows the distribution of the retrieval times: For 143 classes of all the queried classes, SCREECH-ALL has 50% time saving while it is 75% for 95 classes. A summary of the results can be seen in Table 2. For 137 of the 175 classes (i.e. 78%), the computed extensions under SCREECH-ALL and SCREECH-NONE had the same number of elements, which allows to conclude – without using the disjunctive rules – that for those classes the extensions were computed correctly. For some classes, so for the class `Physical-occurrent-entity`, computing the extension under SCREECH-ALL saved 99% of the runtime.

While the different versions of SCREECH have about the same runtime, the differences in the number of introduced errors is remarkable. Indeed, SCREECH-NONE makes almost no mistakes. The parallel execution of SCREECH-NONE and

[10] The exact numbers differ slightly on different runs, as the KAON2 translation algorithm is non-deterministic. Here it was between 1737 and 1909.

[11] The number of disjunctive rules ranged between 51 and 81 on different runs.

Table 3. Summary of the three SCREECH versions on DOLCE. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
SCREECH-ALL	0	3697	2256	472941	516064	0.766	0.76	0.92
SCREECH-ONE	0	3697	512	425748	516064	0.935	1.0	0.82
SCREECH-NONE	0	3697	0	397260	516064	1.0	1.0	0.77

SCREECH-ALL, as mentioned, allows to compute the correct extensions of 78% of the classes – and to know that the computations are correct – in less than a quarter of the time needed by using the unmodified knowledge base.

4.3 DOLCE

DOLCE[12] (a Descriptive Ontology for Linguistic and Cognitive Engineering) is a foundational ontology, developed by the Laboratory for Applied Ontology in the Institute of Cognitive Sciences and Technology of the Italian National Research Council. In full, it exceeds the reasoning capabilities of current reasoners, hence we used a fraction for our experiments consisting of 1552 axioms. Since DOLCE is a pure TBox-Ontology, we randomly populated it with 502 individuals to be able to carry out instance retrieval.

The conversion into disjunctive datalog yielded ca. 1774 rules[13] of which ca. 71 are disjunctive.[14] The SCREECH-ALL split resulted in 178 new rules, replacing the disjunctive ones. We also removed ca. 189 integrity constraints.[15]

As before, we queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits. Table 3 summarizes. In SCREECH-ALL, 93 of the 123 classes (i.e. 76%) are correctly queried, while in SCREECH-ONE 100 classes are correctly queried.

Remarkable under DOLCE is that SCREECH-NONE makes no mistakes, while the runtime improvement is rather mild. This indicates that the disjunctive knowledge in DOLCE does not contribute any results.

4.4 WINE

The next ontology we tested was the WINE ontology.[16] It is a well-known ontology containing a classification of wines. Moreover, it is one of the rare ontologies with both an ABox and a nontrivial TBox. It also contains nominals, which we removed in an approximate way following [10].[17] The resulting ontology contains

[12] http://www.loa-cnr.it/DOLCE.html
[13] 1774–1788.
[14] 71–73.
[15] 188–190.
[16] http://www.schemaweb.info/schema/SchemaDetails.aspx?id=62
[17] We used the TBox *after* that processing as baseline, since we are interested in the comparion of the different versions of SCREECH.

Table 4. Summary of the three SCREECH versions on WINE. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
SCREECH-ALL	0	30627	1353	588562	707476	0.978	0.93	0.83
SCREECH-ONE	0	30627	615	494456	707476	0.990	0.94	0.70
SCREECH-NONE	697	29930	0	504914	707476	0.988	0.90	0.71

20762 axioms, including functionality, disjunctions, and existential quantifiers. The corresponding ABox contains 6601 axioms.

The translation procedure into disjunctive datalog produces altogether ca. 554 rules,[18] among them 24 disjunctive ones. The SCREECH-ALL split resulted in 48 new rules, replacing the disjunctive ones. We removed 3 integrity constraints after the translation.

As before, we queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits. A summary of the results can be seen in Table 4. For 130 of the 140 classes (83%), under SCREECH-ALL we obtained 1353 incorrect extensions, while under SCREECH-ONE 132 classes are correct queried. Under SCREECH-NONE, the number of the classes correctly queried is 126, and totally 697 extensions were missing.

WINE is the only ontology we tested for which SCREECH-NONE resulted in mildly significant number of mistakes. However, recall is still at 0.977, i.e. very good. Considering the fact that WINE was created to show the expressiveness of OWL DL, it is remarkable that all three SCREECH versions show a very low amount of errors, while runtime increases by 28.6–34.5%. For some classes – e.g. for `Chianti`, over 91% of the runtime was saved using SCREECH-ALL.

4.5 SEMINTEC

We also considered an ontology, the translation of which turned out to not contain proper disjunctive rules. Nevertheless, removing integrity constraints is supposed to result in improving runtime behaviour (while in this case even preserving soundness).

So, the last ontology we considered is from the SEMINTEC project[19] at the university of Poznan and concerns financial services. Its TBox contains 130702 axioms of comparably simple structure, apart from some functionality constraints which require equality reasoning. The ABox contains 71764 axioms. The TBox translation generated 217 rules, all of them being Horn, among which were 113 integrity constraints.

As before, we queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits. A summary of the results can be seen in Table 5. As in the case

[18] 526–572.

[19] `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`

Table 5. Summary of Screech on SEMINTEC – note that all three versions of Screech coincide, since no disjuntive rules are produced by the translation. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
Screecha-all	0	51184	0	31353	94981	1.0	1.0	0.33
Screech-one	0	51184	0	32200	94981	1.0	1.0	0.33
Screech-none	0	51184	0	32032	94981	1.0	1.0	0.33

Table 6. Summary of Screech on VICODI – note that all three versions of Screech coincide, since no disjuntive rules are produced by the translation. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
Screech-all	0	282564	0	3228	7192	1.0	1.0	0.45
Screech-one	0	282564	0	3295	7192	1.0	1.0	0.46
Screech-none	0	282564	0	3346	7192	1.0	1.0	0.47

Table 7. Overview of Screech evaluations. Mark that for due to the completeness of Screech-all, the recall values are always 100% as well as the precision values for Screech-none due to its soundness. Moreover, the three Screech variants coincide in the case of the SEMINTEC ontology.

	Screech-all			Screech-one			Screech-none		
ontology	time saved	precision	recall	time saved	precision	recall	time saved	precision	recall
GALEN	74.6%	91.7%	100%	72.5%	97.4%	99.9%	76.5%	100%	99.8%
DOLCE	29.1%	62.1%	100%	17.5%	87.8%	100%	23.0%	100%	100%
WINE	34.5%	95.8%	100%	30.1%	98.0%	100%	28.6%	100%	97.7%
SEMINTEC	69.9%	100%	100%	66.0%	100%	100%	66.2%	100%	100%
VICODI	55.1%	100%	100%	54.1%	100%	100%	53.4%	100%	100%

of absence of disjunctive rules all three variants of Screech coincide, for all of the 60 classes, the extensions were computed correctly.

For SEMINTEC, we achieve a performance improvement of 54% while the computation remains correct. For some classes – in particular for some with very small extensions, computing the extension under Screech-all saved about 95% of the runtime. For some classes with larger extension – like `Leasing`, 92% of runtime was saved.

4.6 VICODI

Another ontology containing no disjunctive rules is the VICODI ontology. But it has a large ABox. It also has no integrity constraints. Hence, the knowledge bases generated are the same; as given in Table 3 they resulted in a datalog program with 223 Horn rules. Like SEMINTEC, for all of the 194 classes, the extensions were computed correctly in each Screech variant. The performance gain for Screech-all, Screech-one and Screech-none is 55.1%, 54.4% and 53.4%, respectively.

5 Conclusions

Motivated by the obvious need for techniques enhancing the scalability of reasoning related tasks, we have investigated three variants of the Screech approach to approximate reasoning in OWL ontologies.

On the theoretical side, we gave the completeness result for Screech-all and the soundness result for Screech-none, yet a desirable characterisation of the approximations in terms of extremal bounds following the theory of Horn-approximations was shown not to hold by providing counterexamples.

However, on the practical side the obtained results were promising: the performance improvement is stable over all ontologies which we included in our experiments. The performance gain varied between 29.1 and 76.5%, while the amount of correctly retrieved classes was above 76% for all but one of the ontologies – see Table 7. It is encouraging to see that the approach appears to be feasible even for the sophisticated WINE ontology, and also for the SEMINTEC ontology, although in the latter case we only remove integrity constraints. Concerning the comparatively bad results on DOLCE, we note that the results are quite counterintuitive. One would naively expect that performance improvements go hand-in-hand with loss of precision. However, for DOLCE we measured both the least runtime improvement and the worst performance in terms of correctness. Concerning correctness, we suspect that the comparatively large number of incorrect answers is caused by the fact that DOLCE uses a large number of complete partitions of the form $A \equiv A_1 \sqcup \cdots \sqcup A_n$, where all the A_i are also specified to be mutually disjoint. It is intuitively clear that this kind of axioms introduces disjunctive (non-Horn-style and therefore harder to approximate) information on the one hand and integrity constraints (those being neglected in our approximation) on the other. However, this does not in itself explain why we did not observe a higher speedup. This indicates that the properties of ontologies which lead to performance improvement through screeching must be less straightforward than initially expected. For a clarification, more evaluations taking into account a wider range of ontologies with differing characteristics w.r.t. expressivity, used language features, or statistical measures like degree of population will lead to substantial hypotheses.

In general, we see a great potential in the strategy to combine various (possibly approximate) algorithms having known properties as soundness and/or completeness maybe with respect to differing types of queries. For instance, the proposed "sandwich technique" can be used to solve instance retrieval tasks in some cases even without calling the more costly sound and complete reasoners. If the sets of individuals I_S and I_C retrieved by two algorithms—one of those being sound and the other one complete—coincide, the result is known to be exact. But even if not, the result is at least partly determined (as all elements of I_S are definitely instances and all individuals *not* in I_C are not) and it might be beneficial to query a sound and complete reasoner for class membership only for individuals of the set $I_C \setminus I_S$ of individuals for which class membership is still undecided. Clearly, the strategy to combine several approximate algorithms will be especially reasonable if parallel computation architectures are available.

References

1. McGuinness, D.L., van Harmelen, F.: OWL web ontology language overview (2004), http://www.w3.org/TR/owl-features/
2. Grosof, B., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logics. In: Proc. of WWW 2003, Budapest, Hungary, May 2003, pp. 48–57. ACM, New York (2003)
3. Volz, R.: Web Ontology Reasoning with Logic Databases. PhD thesis, Institute AIFB, University of Karlsruhe (2004)
4. Krötzsch, M., Hitzler, P., Vrandecic, D., Sintek, M.: How to reason with OWL in a logic programming system. In: Eiter, T., Franconi, E., Hodgson, R., Stephens, S. (eds.) Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web, RuleML2006, Athens, Georgia, pp. 17–26. IEEE Computer Society, Los Alamitos (2006)
5. Krötzsch, M., Rudolph, S., Hitzler, P.: Description Logic Rules. In: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008). IOS Press, Amsterdam (to appear, 2008)
6. Krötzsch, M., Rudolph, S., Hitzler, P.: Elp: Tractable rules for OWL 2. In: Proceedings of the 7th International Semantic Web Conference (ISWC 2008) (to appear, 2008)
7. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, pp. 466–471 (2005)
8. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Universität Karlsruhe (2006)
9. Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic ABoxes. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246. Springer, Heidelberg (2006)
10. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In: Gil, Y., et al. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 383–397. Springer, Heidelberg (2005)
11. Rudolph, S., Krötzsch, M., Hitzler, P., Sintek, M., Vrandecic, D.: Efficient OWL reasoning with logic programs – evaluations. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 370–373. Springer, Heidelberg (2007)
12. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM 42, 741–843 (1995)
13. Fensel, D., Harmelen, F.V.: Unifying reasoning and search to web scale. IEEE Internet Computing 11, 94–96 (2007)
14. Stuckenschmidt, H., van Harmelen, F.: Approximating terminological queries. In: Larsen, H., et al. (eds.) FQAS 2002. LNCS (LNAI), vol. 2522. Springer, Heidelberg (2002)

15. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL reasoning with large numbers of individuals. In: Proceedings of the International Workshop on Description Logics, DL 2004, Whistler, Canada, pp. 31–40 (2004)
16. Groot, P., Stuckenschmidt, H., Wache, H.: Approximating description logic classification for semantic web reasoning. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 318–332. Springer, Heidelberg (2005)
17. Wache, H., Groot, P., Stuckenschmidt, H.: Scalable instance retrieval for the semantic web by approximation. In: Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z. (eds.) WISE 2005 Workshops. LNCS, vol. 3807, pp. 245–254. Springer, Heidelberg (2005)
18. Pan, J.Z., Thomas, E.: Approximating OWL-DL ontologies. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 22-26, 2007, pp. 1434–1439. AAAI Press, Menlo Park (2007)
19. Stuckenschmidt, H.: Partial matchmaking using approximate subsumption. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 22-26, 2007, pp. 1459–1464. AAAI Press, Menlo Park (2007)
20. Sakama, C., Inoue, K.: An alternative approach to the semantics of disjunctive logic programs and deductive databases. Journal of Automated Reasoning 13, 145–172 (1994)
21. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM 42, 741–843 (1995)
22. Angele, J., Lausen, G.: Ontologies in F-logic. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 29–50. Springer, Heidelberg (2004)
23. Selman, B., Kautz, H.A.: Knowledge compilation using horn approximations. In: Proc. 9th National Conference on Artificial Intelligence (AAAI 1991), pp. 904–909 (1991)

Ranking Services Using Fuzzy HEX Programs[⋆]

Stijn Heymans[1,2] and Ioan Toma[3]

[1] Knowledge-Based Systems Group, Institute of Information Systems, Vienna University of Technology, Austria
heymans@kr.tuwien.ac.at
[2] Computational Web Intelligence, Department of Applied Mathematics and Computer Science, Ghent University, Belgium
[3] Semantic Technology Institute (STI) Innsbruck, University of Innsbruck, Austria
ioan.toma@sti-innsbruck.at

Abstract. The need to reason with knowledge expressed in both Logic Programming (LP) and Description Logics (DLs) paradigms on the Semantic Web lead to several integrating formalisms, e.g., Description Logic programs (*dl-programs*) allow a logic program to retrieve results from and feed results to a DL knowledge base. Two functional extensions of dl-programs are HEX programs and fuzzy dl-programs. The former abstract away from DLs, allowing for general external queries, the latter deal with the uncertain, vague, and inconsistent nature of knowledge on the Web by means of fuzzy logic mechanisms. In this paper, we generalize both HEX programs and fuzzy dl-programs to *fuzzy* HEX *programs*: a LP-based paradigm, supporting both fuzziness as well as reasoning with external sources. We define basic syntax and semantics and analyze the framework semantically, e.g., by investigating the complexity. Additionally, we provide a translation from fuzzy HEX programs to HEX programs, enabling an implementation via the dlvhex reasoner. Finally, we illustrate the use of fuzzy HEX programs for ranking services by using them to model non-functional properties of services and user preferences.

1 Introduction

Logic Programming (LP) [2] and Description Logics (DLs) [1] are two of the main underlying knowledge representation and reasoning paradigms of the Semantic Web, a machine-understandable instead of just machine-readable Web [4]. Logic Programming underlies, for example, several variants of the Web Service Modeling Language WSML [6] and Description Logics form the basis of the ontology language OWL-DL [3].

As the Semantic Web is about understanding knowledge and automatizing inferences from this knowledge, it is not surprising that there is a lot of interest

⋆ Stijn Heymans is partially supported by the Austrian Science Fund (FWF) under project P20305-N18 and the Fund for Scientific Research Flanders (FWO Vlaanderen) under project 3G010107. Ioan Toma is supported by the EU FP7 IST project 27867, *SOA4ALL - Service Oriented Architectures For All.*

in the integration of these paradigms (see, e.g., [5] for an overview). One of these integrating approaches are *Description Logic Programs*, dubbed *dl-programs* [11], that take a LP view on a DL knowledge base: logic programs are able to query DL knowledge bases via *dl-atoms*. Moreover, dl-atoms can stream knowledge from the logic program to the DL knowledge base, where it can be used to make additional DL inferences (which can then in turn be used in the LP deduction process). In effect, there is a bi-directional stream of information between the logic program and the DL knowledge base.

In [10], dl-programs were generalized to HEX *programs*. HEX programs combine higher-order reasoning - naively put, they allow for variables to appear in the predicate position, enabling thus meta-reasoning over concepts - and external atoms. The latter generalize dl-atoms as they do not just access DL knowledge bases but are associated with any external function - one can use them, e.g., to query RDF repositories or SQL databases.

Another extension to dl-programs was inspired by the uncertainty, vagueness, and inconsistency of the (Semantic) Web. As anyone can produce knowledge on the Web, it is impossible to ensure that all knowledge on the Web is logically true. Moreover, often there is a need (as there is in real-life) to express vague concepts, such as *very*, *beautiful*, or *old/young*; a need that is not met by traditional two-valued logics like LP or DLs. And finally, the Web is inconsistent: source A might have another (contradicting) opinion than source B on a topic. Together with the need for integrating approaches, this lead to so-called *fuzzy dl-programs* [14,12]. Fuzzy dl-programs extend dl-programs by allowing to query fuzzy DLs [17] and by using fuzzy dl-rules on the LP side.

Intuitively, fuzzy dl-rules use *combination strategies* instead of the usual conjunction, disjunction, and negation in normal LP rules. Those combination strategies do nothing else than computing a resulting truth value based on two (or one, in the case of the negation strategy) input truth values, where truth values, in contrast with two-valued logics, range over the interval $[0, 1]$. For example, in normal LP, a rule $fail \leftarrow not\ study, smart$ where *study* is false (or 0) and *smart* is true, results in a truth value of 1 for *fail*. A fuzzy variant could be $fail \leftarrow_{\otimes_G} not_{\ominus_L}\ study \otimes_G smart \geq 0.5$ where $\otimes_G$ is the Gödel conjunction (which takes the minimum of two values) and $\ominus_L$ is the Lukasiewizc negation (which takes the complement of a value w.r.t. 1). If *study* has a fuzzy value of 0.4 and *smart* of 0.9, we would have that $not_{\ominus_L}\ study$ has a value of 0.6. The value 0.5 indicates to what extent the rule should be satisfied. Using $\otimes_G$ we would have that the value of the body (the part to the right of $\leftarrow$) has to be $0.5 \otimes_G 0.6 \otimes_G 0.9 = 0.5$ and that the value of *fail* should be at least this value (0.5) in order to make the fuzzy rule satisfied. Note that the value 0.5 that indicates to which degree a rule should be satisfied is used to calculate the value of the body.

In this paper, we generalize both extensions - from dl-programs to HEX programs and from dl-programs to fuzzy dl-programs - to *fuzzy* HEX *programs*. Fuzzy HEX programs thus support higher-order reasoning, reasoning with external sources like DL knowledge bases or in general with external functions (e.g., *sum*, *max*), and fuzzy reasoning. We establish the basic syntax and semantics of such programs and show that the complexity of disjunction-free fuzzy HEX programs, under equal conditions for the external predicates and appropriately behaving fuzzy combination strategies, is the same as for disjunction-free HEX programs, namely NEXPTIME-complete.

We furthermore establish a translation from fuzzy HEX programs to HEX programs, basically writing the combination strategies as external predicates that can be computed by external functions. This enables reasoning with fuzzy HEX programs using the DLVHEX [9] reasoner for HEX programs.

To show the applicability of fuzzy HEX programs, we use them to describe non-functional properties of Web services, enabling better ranking of services. A *service* is a provision of value to a client [15], e.g., the delivery of a package with some specified constraints. *Service ranking* is then the process which generates an ordered list of services out of the candidate services set according to user's preferences. As ranking criteria, specified by the user, various aspects of a service description can be used. We differentiate between (1) *functional*, (2) *behavioral*, and (3) *non-functional*. The *functional* description contains the formal specification of what exactly the service can do. The *behavioral* description is about how the functionality of the service can be achieved in terms of interaction with the service as well as in terms of functionality required from other services. Finally, the *non-functional description* captures constraints over the previous two [7]. For example, in case of a shipping service, invoking its functionality (shipping a package) might be constrained by paying a certain amount (*price* as non-functional property).

As part of our previous work [18], we have proposed an approach for the service ranking problem based on the evaluation of non-functional properties such as price, response time, liability, etc. Rules encoding conditions and constraints over multiple non-functional properties are used to model both users and service provides perspectives. Although this modeling approach is useful for modeling some of the multitude of non-functional properties such as liability/contractual obligations, for properties, like price or delivery time, a more natural choice to express requests and preferences requires a formalism for handling vagueness and imprecision, e.g., imagine a provider advertising "the cheapest service". Another issue is that service descriptions, besides fuzzy information, often need to refer to external libraries and data sources. Fuzzy HEX programs address exactly these requirements.

The remainder of the paper starts with an introduction to HEX programs in Section 2. Section 3 defines fuzzy HEX programs with its basic properties. In Section 4, we show that fuzzy HEX programs generalize both HEX programs and fuzzy dl-programs, and Section 5 gives complexity results for fuzzy HEX programs as well as a translation from fuzzy HEX programs to HEX programs, allowing an implementation using DLVHEX. Section 6 contains an application of fuzzy HEX programs to the problem of ranking services and we give directions for further research in Section 7. Proofs of the key results can be found at `http://www.kr.tuwien.ac.at/staff/heymans/fuzzy-hex-proofs.pdf`.

2 Preliminaries: HEX Programs

We introduce HEX programs as in [10]. Assume the existence of 3 mutually disjoint sets $\mathcal{C}, \mathcal{X}$, and $\mathcal{G}$, consisting of constants, variables, and external predicates respectively. A *term* is either a *constant* $a \in \mathcal{C}$ or a *variable* $X \in \mathcal{X}$, denoted with symbols starting with lower-case or upper-case letters respectively. A *higher-order atom* is of the form $t_0(t_1, \ldots, t_n)$ for terms t_i, $0 \leq i \leq n$. If t_0 is a constant, we call $t_0(t_1, \ldots, t_n)$ an *ordinary atom*. An *external predicate* from $\mathcal{G}$ starts with the symbol #, e.g., $\#g$ or $\#sqrt$, where each external predicate has an associated *input* and *output arity*. An *external atom* is of the form $\#g[t_1, \ldots, t_n](s_1, \ldots, s_m)$ where $t_1, \ldots, t_n$ is the *input list* of terms for the input arity n of $\#g$ and $s_1, \ldots, s_m$ is the *output list* of terms for the output arity m of $\#g$.

A *rule* r is of the form:

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m \tag{1}$$

where $a_1, \ldots, a_k$ are higher-order atoms, and $b_1, \ldots, b_m$ are higher-order or external atoms. The *head* of r is $\text{head}(r) = \{a_1, \ldots, a_k\}$, and the *body* of r is $\text{body}(r) = \text{body}^+(r) \cup \text{body}^-(r)$ with $\text{body}^+(r) = \{b_1, \ldots, b_n\}$ and $\text{body}^-(r) = \{b_{n+1}, \ldots, b_m\}$. A rule is ordinary if it only contains ordinary atoms. If $k = 1$ we call the rule *disjunction-free*. A *(disjunction-free)* HEX *program* is a finite set P of (disjunction-free) rules.

An atom (higher-order or external), rule, or program, is *ground* if no variables appear in it. A *grounding* of a program P is a ground program $gr(P)$ that contains all possible ground rules resulting from replacing the variables in those rules with all possible constants from $\mathcal{C}$. The *Herbrand Base* of P, denoted $\mathcal{B}_P$, is the set of all possible ground versions of atoms (higher-order or external) occuring in P using constants of $\mathcal{C}$. Note that the Herbrand Base only contains ordinary atoms and external atoms. If $\mathcal{C}$, $\mathcal{X}$, or $\mathcal{G}$ are not explicitly given, we assume they are implicitly given by the program P under consideration.

An *interpretation* I of a program P is a set $I \subseteq \mathcal{B}_P$ of ordinary atoms (i.e., no external atoms). We say that I is a *model* of a ground ordinary atom a, denoted $I \models a$, if $a \in I$.

We associate with every external predicate symbol $\#g \in \mathcal{G}$, an $(n+m+1)$-ary function $f_{\#g}$, that assigns a tuple $(I, y_1, \ldots, y_n, x_1, \ldots, x_m)$ to 0 or 1, with n the input arity of $\#g$ and m the output arity of $\#g$, I an interpretation, and y_i, x_j constants. I is then a *model* of a ground external atom $a = \#g[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, if and only if $f_{\#g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$. For a ground atom (possibly external) a, we have $I \models not\ a$ iff $I \not\models a$. This definition extends for sets containing ground ordinary and ground external atoms as usual.

We say that a ground rule r is *satisfied* by I, denoted $I \models r$, if, whenever $I \models \text{body}^+(r) \cup not\ \text{body}^-(r)$, we have that there is a a_i, $1 \leq i \leq k$, such that $I \models a_i$. For a HEX program P, I is a *model* of P, denoted $I \models P$, iff $I \models r$ for each $r \in gr(P)$. We define the *FLP-reduct* P^I of a program w.r.t an interpretation I as all rules $r \in gr(P)$ such that $I \models \text{body}^+(r) \cup not\ \text{body}^-(r)$. An interpretation I of P is an *answer set* of P iff I is a minimal model of P^I.

3 Fuzzy HEX Programs

We use the definition of fuzzy *dl-rules* from [14,12] as an inspiration to extend HEX programs to fuzzy HEX programs, and we start by identifying different *combination strategies*:

- The *negation* strategy $\ominus : [0,1] \to [0,1]$, where we call $\ominus v$, $v \in [0,1]$, the *negation* of v. The negation strategy has to be *antitonic*, i.e., if $v_1 \leq v_2$, then $\ominus v_1 \geq \ominus v_2$. Furthermore, we have that $\ominus 1 = 0$ and $\ominus 0 = 1$. Particular negation strategies ([17]) are, for example, the *Lukasiewizc negation* $\ominus_L$, defined by $\ominus_L x = 1 - x$, or the *Gödel negation* $\ominus_G$, defined by $\ominus 0 = 1$ and $\ominus x = 0$ if $x > 0$.
- The *conjunction* strategy $\otimes : [0,1] \times [0,1] \to [0,1]$, where we call $v_1 \otimes v_2$, $v_1, v_2 \in [0,1]$, the conjunction of v_1 and v_2. The conjunction strategy has to be *commutative*, *associative*, and *monotone* (if $v_1 \leq v_1'$ and $v_2 \leq v_2'$, then $v_1 \otimes v_2 \leq v_1' \otimes v_2'$). Furthermore, we need to have that $v \otimes 1 = v$ and $v \otimes 0 = 0$. Particular conjunction strategies (also called *t-norms* [17]) are, for example, the *Lukasiewizc conjunction* $\otimes_L$, defined by $x \otimes_L y = \max(x + y - 1, 0)$, the *Gödel conjunction* $\otimes_G$, defined by $x \otimes_G y = \min(x, y)$, and the *product conjunction* $\otimes_P$, defined by $x \otimes_P y = x.y$.
- The *disjunction* strategy $\oplus : [0,1] \times [0,1] \to [0,1]$, where we call $v_1 \oplus v_2$, $v_1, v_2 \in [0,1]$, the disjunction of v_1 and v_2. The disjunction strategy has to be *commutative*, *associative*, and *monotone* (if $v_1 \leq v_1'$ and $v_2 \leq v_2'$, then $v_1 \oplus v_2 \leq v_1' \oplus v_2'$). Furthermore, we need to have that $v \oplus 1 = 1$ and $v \oplus 0 = v$. Particular disjunction strategies (also called *s-norms* [17]) are, for example, the *Lukasiewizc disjunction* $\oplus_L$, defined by $x \oplus_L y = \min(x + y, 1)$, the *Gödel disjunction* $\oplus_G$, defined by $x \oplus_G y = \max(x, y)$, and the *product disjunction* $\oplus_P$, defined by $x \oplus_P y = x + y - x.y$.

Definition 1. *A* fuzzy rule *r is of the form*

$$\begin{aligned} a_1 \oplus_1 \ldots \oplus_{k-1} a_k \ \leftarrow_{\otimes_0} \ & b_1 \otimes_1 \ldots \otimes_{n-1} b_n \\ & \otimes_n not_{\ominus_{n+1}} \ b_{n+1} \otimes_{n+1} \ldots \otimes_{m-1} not_{\ominus_m} \ b_m \geq v \end{aligned} \quad (2)$$

where $a_1, \ldots, a_k$ are higher-order atoms, $b_1, \ldots, b_m$ are higher-order, external atoms, or elements from $[0,1]$, and $v \in [0,1]$. The head *and* body *of r is defined as before. A* (disjunction-free) fuzzy HEX program *is a finite set P of (disjunction-free) fuzzy rules.*

Note that a fuzzy rule can contain different negation strategies; the order of evaluation of such strategies will be *left-to-right*.

Ground atoms, rules, programs, as well as a *grounding* are defined similarly as for HEX programs.

A *fuzzy interpretation* of a fuzzy HEX program is a mapping $I : O_P \subseteq \mathcal{B}_P \to [0,1]$ where O_P are the ordinary atoms in $\mathcal{B}_P$. Define $I \subseteq J$ for fuzzy interpretations I and J of P, if $I(a) \leq J(a)$ for each $a \in O_P$. We call I *minimal* if there is no interpretation $J \neq I$ such that $J \subseteq I$. The *fuzzy value* v_I of a ground ordinary atom a w.r.t. an interpretation I is $v_I(a) = I(a)$.

We associate with every external predicate symbol $\#g \in \mathcal{G}$, an $(n+m+1)$-ary function $f_{\#g}$, that assigns a tuple $(I, y_1, \ldots, y_n, x_1, \ldots, x_m)$ to $[0,1]$, with n the input arity of $\#g$ and m the output arity of $\#g$, I a fuzzy interpretation, and y_i, x_j constants. The *fuzzy value* v_I of a ground external atom $a = \#g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ w.r.t. an interpretation I is $v_I(a) = f_{\#g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m)$. We complete the definition of v_I by defining it for values v from $[0,1]$ as $v_I(v) = v$.

A fuzzy interpretation I *satisfies* a ground fuzzy rule (2) iff

$$\begin{aligned} v_I(a_1) \oplus_1 \ldots \oplus_{k-1} v_I(a_k) \ &\geq v \otimes_0 v_I(b_1) \otimes_1 \ldots \otimes_{n-1} v_I(b_n) \\ &\otimes_n \ominus_{n+1} v_I(b_{n+1}) \otimes_{n+1} \ldots \otimes_{m-1} \ominus_m v_I(b_m) \ . \end{aligned} \tag{3}$$

A fuzzy interpretation I is a *fuzzy model*[1] of a fuzzy HEX program P if it satisfies every rule in $gr(P)$.

The *FLP-reduct* P^I of a fuzzy HEX program w.r.t a fuzzy interpretation I are all rules $r \in gr(P)$ of the form (2) where

$$v \otimes_0 v_I(b_1) \otimes_1 \ldots \otimes_{n-1} v_I(b_n) \otimes_n \ominus_{n+1} v_I(b_{n+1}) \otimes_{n+1} \ldots \otimes_{m-1} \ominus_m v_I(b_m) > 0 \ .$$

We can then define fuzzy answer sets as follows:

Definition 2. *Let P be a fuzzy* HEX *program. A fuzzy interpretation I of P is a* fuzzy answer set *of P iff I is a minimal fuzzy model of P^I.*

Example 1. Take P with rules $a \leftarrow_{\otimes_P} not_{\ominus_L}\ b \geq 1$ and $b \leftarrow_{\otimes_P} not_{\ominus_L}\ a \geq 1$. One can check that a fuzzy interpretation I_1 with $I_1(a) = 0.8$ and $I_1(b) = 0$ is not a model of P and thus not a fuzzy answer set. On the other hand, I_2 with $I_2(a) = 0.6$ and $I_2(b) = 0.4$ is a fuzzy answer set.

Example 2. Take the program P with rule $a \leftarrow_{\otimes_P} not_{\ominus_L}\ a \geq 1$. Although the normal program $a \leftarrow not\ a$ has no answer sets, the fuzzy version of this program has a fuzzy answer set I where $I(a) = \frac{1}{2}$, i.e., if one is equally unsure about a as about *not* a, the contradicting rule is no longer relevant.

A *positive* fuzzy HEX program is a program without negation strategies.

We have that for positive programs the FLP-reduct has no influence on the fuzzy answer sets:

Proposition 1. *Let P be a positive fuzzy* HEX *program. Then, M is a fuzzy answer set of P iff M is a minimal fuzzy model of P.*

Proposition 1 does not necessarily hold if P is not positive as one can see from the fuzzy program $a \leftarrow_{\otimes_P} not_{\ominus}\ a \geq 1$ where we define $\ominus$ as follows: $\ominus x = 1$ for $x < 0.1$ and $x = 0$ for $x \geq 0.1$.

[1] We will omit the modifier *fuzzy* if it is clear from the context.

4 Fuzzy HEX Programs Generalize HEX Programs and Fuzzy dl-Programs

To show that HEX programs are properly embedded in fuzzy HEX programs we introduce a *crisp conjunction* $x \otimes_c y = 1$ if $x = 1 \wedge y = 1$ and 0 else, a *crisp disjunction* $x \oplus_c y = 1$ if $x = 1 \vee y = 1$ and 0 else, and a *crisp negation* $\ominus_c x = 0$ if $x = 1$ and 1 else.

Proposition 2. *The crisp conjunction (disjunction, negation) is a well-defined conjunction (disjunction, negation) strategy.*

For a HEX program P we define its fuzzy version P^f as follows:

Definition 3. *Let P be a* HEX *program. Then, P^f consists of the rules*

$$a_1 \oplus_c \ldots \oplus_c a_k \leftarrow_{\otimes_c} b_1^f \otimes_c \ldots \otimes_c b_n^f \otimes_c not_{\ominus_c}\, b_{n+1}^f \otimes_c \ldots \otimes_c not_{\ominus_c}\, b_m^f \geq 1 \quad (4)$$

for every rule of the form (1) in P, where b_i^f, $1 \leq i \leq m$, is defined such that $b_i^f = b_i$ when b_i is not external, and, if $b_i = \#g[t_1, \ldots, t_n](s_1, \ldots, s_m)$ then $b_i^f = \#g^f[t_1, \ldots, t_n](s_1, \ldots, s_m)$ where $\#g^f$ is associated with the external function $f_{\#g^f}$ that assigns, for any fuzzy interpretation I, the tuple $(I, y_1, \ldots, y_n, x_1, \ldots, x_m)$ to the value $f_{\#g}(I', y_1, \ldots, y_n, x_1, \ldots, x_m)$ where $a \in I'$ iff $I(a) = 1$, $a \in O_P$.

Proposition 3. *Let P be a* HEX *program. Then, M is an answer set of P iff M^f is a fuzzy answer set of P^f where $M^f : O_P \rightarrow [0,1]$ is such that $M^f(a) = 1$ if $a \in M$ and $M^f(a) = 0$ otherwise.*

Proposition 3 shows that fuzzy HEX programs are layered on HEX programs.

Description Logic Programs (*dl-programs* for short) [11] is a formalism that allows to combine DL knowledge bases with logic programs. Roughly, in a dl-program the logic program can query the DL knowledge base, while possibly feeding deductions from the logic program as input to it. As dl-programs can be embedded in HEX programs [10], and the fuzzy rules we consider are syntactically and semantically similar in spirit as the fuzzy rules used in [12], it comes as no surprise that the so-called *fuzzy dl-programs* from [12] can be embedded in fuzzy HEX programs.

We briefly introduce fuzzy dl-programs and refer the reader for more details to [12]. A *fuzzy dl-program* (L, P) consists of a *fuzzy description logic knowledge base* L and a finite set of ground fuzzy rules P. We again refer to [12] for more details on fuzzy DLs, and retain from [12] that L comes associated with a *models* operator $\models$ such that one can express statements $L \models C(t) \geq v$ for a concept C and a term t and statements $L \models R(t_1, t_2) \geq v$ for a role R and terms t_1 and t_2; v is a value from some $[0,1]$[2]. Intuitively, one can deduce statements from L

[2] Note that [12] restricts itself to a set $TV_n = \{0, \frac{1}{n}, \ldots, 1\}$. We will later restrict ourselves also to this set instead of considering $[0,1]$, but for showing that fuzzy dl-programs are embedded in fuzzy HEX programs we can safely take the more general interval $[0,1]$.

that indicate to what fuzzy degree v, the term t belongs to the concept C (or (t_1, t_2) belongs to R).

Fuzzy dl-rules in P are of the form ([2]) with the following modifications:

– No non-ordinary higher-order or external atoms appear in P,
– Atoms may also be *dl-atoms* $DL[S_1 \cup p_1, \ldots, S_n \cup p_n; Q](d)$, where S_i are concepts or roles, p_i are unary or binary predicates (unary if S_i is a concept, binary if S_i is a role), Q and d are either a concept and a term or a role and a pair of terms. .

For a fuzzy interpretation I of P, the value $v_I(a)$ of a ground dl-atom $a = DL[S_1 \cup p_1, \ldots, S_n \cup p_n; Q](d)$ w.r.t. L is defined as the maximum value $v \in [0,1]$ such that $L \cup \bigcup_{i=1}^{m} A_i(I) \models Q(d) \geq v$ with $A_i(I) = \{S_i(e_i) \geq I(p_i(e_i)) \mid I(p_i(e_i)) > 0\}$ where e_i is a constant or a pair of constants depending on the arity of p_i. Intuitively, we query the DL knowledge base L where the fuzzy degrees of the concepts/roles S_i are augmented with what we know from P (i.e., via the p_i predicates) to find out what the fuzzy degree v is of membership of d in Q.

A fuzzy answer set of such a fuzzy dl-program is then defined analogous to our fuzzy answer sets where dl-atoms a have the value $v_I(a)$ w.r.t. L as defined above.

Similar as in [10], we can replace dl-atoms $a = DL[S_1 \cup p_1, \ldots, S_n \cup p_n; Q](d)$ by external atoms $\#a_L[\,](d)$ such that the associated external function $f_{\#a_L}(I, d) = v$ iff $L \cup \bigcup_{i=1}^{m} A_i(I) \models Q(d) \geq v$. For a fuzzy dl-program (L, P), let $P^{\#}$ be the program obtained from P by replacing all dl-atoms $a = DL[S_1 \cup p_1, \ldots, S_n \cup p_n; Q](d)$ by their external version $\#a_L[\,](d)$.

Proposition 4. *Let (L, P) be a fuzzy dl-program. Then, M is a fuzzy answer set of (L, P) iff M is a fuzzy answer set of $P^{\#}$.*

5 Complexity and Reasoning

We restrict ourselves in the following to the fixed set $TV_n = \{0, \frac{1}{n}, \frac{2}{n}, \ldots, 1\}$ instead of the interval $[0,1]$, and we assume, similar as in [13], that the combination strategies are *closed* in TV_n. Note that the Lukasiewizc and Gödel combination strategies are all closed, but, for example, the production conjunction is not: on TV_3 we have that $\frac{1}{3} \otimes_P \frac{2}{3} = \frac{2}{9} \notin TV_3$. Additionally, we assume that external functions $f_{\#}g$, associated with external predicates $\#g$, are defined as functions $f_{\#g} : TV_n \to TV_n$.

For combination strategies $\otimes$ and $\ominus$, we assume the existence of external atoms $\#\otimes[X, Y](Z)$ and $\#\ominus[X](Z)$, with associated external functions to $\{0, 1\}$ defined as follows for a fuzzy interpretation I: $f_{\#\otimes}(I, X, Y, Z) = 1$ iff $X \otimes Y = Z$ and $f_{\#\ominus}(I, X, Z) = 1$ iff $\ominus X = Z$.

Additionally, we define a $\#max[X](Y)$ atom such that $f_{\#max}(I, X, Y) = 1$ if $Y = \max\{v \mid X(v) \in I\}$, i.e., Y is the maximum value that the argument of an X-atom can take.

We transform a disjunction-free fuzzy HEX program P in a HEX program P^h:

Definition 4. *Let P be a disjunction-free fuzzy* HEX *program. We take $\mathcal{C} = TV_n$. Then P^h consists of rules*

$$\sigma_a(x) \leftarrow \tag{5}$$

for each non-external atom $a \in gr(P)$[3] *where $x = a$ if $a \in TV_n$ and $x = 0$ otherwise, rules*

$$\sigma_a(X) \leftarrow \#g^h[y_1, \dots, y_n](x_1, \dots, x_m, X) \tag{6}$$

for each external atom $a = \#g[y_1, \dots, y_n](x_1, \dots, x_m) \in gr(P)$, where

$$f_{\#g^h}(I^h, y_1, \dots, y_n, x_1, \dots, x_m, x) = 1 \text{ iff } f_{\#g}(I, y_1, \dots, y_n, x_1, \dots, x_m) = x \ ,$$

for any interpretation I^h of P^h and I its fuzzy variant *defined such that, for a non-external atom a, $I(a) = \max\{y \mid \sigma_a(y) \in I^h\}$, and for each rule with non-empty body of the form (2) in $gr(P)$, rules*

$$\begin{aligned}\sigma_a(U_m) \ \leftarrow \ & \\ & \sigma_{b_1}(X_1), \#max[\sigma_{b_1}](X_1), \dots, \\ & \sigma_{b_m}(Y_m), \#max[\sigma_{b_m}](Y_m), \#\ominus_m[Y_m](X_m), \#\otimes_0[U_{m-1}, v](U_m), \\ & \#\otimes_1[X_1, X_2](U_1), \#\otimes_2[U_1, X_3](U_2), \dots, \#\otimes_{n-1}[U_{m-2}, X_m](U_{m-1})\end{aligned} \tag{7}$$

for rules with empty body, we introduce $\sigma_a(U_m) \leftarrow \#\otimes_0[1, v](U_m)$, and finally rules

$$\sigma_a(x - \frac{1}{n}) \leftarrow \sigma_a(x) \tag{8}$$

for all non-external atoms $a \in gr(P)$ and for all $x \neq 0 \in TV_n$.

Intuitively, the rules (5) make sure that the initial fuzzy value of a non-external atom a is equal to its fuzzy value if $a \in TV_n$ or 0 otherwise, where the value of an atom a is encoded using the binary predicate σ_a. Note that atoms from P are treated as constants in P^h (that are, however, not used to ground the transformed program, see the defintion of $\mathcal{C}$ for P^h). Similarly, the rules in (6) ensure that the values of the external atoms are correctly set. The rules in (7) compute the value of the head atom a based on the maximum values of its body atom, i.e., we assume implicitly that the actual fuzzy value of an atom is the maximum value that is present in the interpretation for that atom (using again the σ-encoding for values). We use the external atoms that correspond to the combination strategies to compute the value of the body and impose that the value of the body is equal to the value of the head, namely U_m. Note that for satisfaction of fuzzy rules the value of the head just needs to be greater than or equal the value of the body; we impose equality to ensure minimality of the fuzzy interpretation. The case where the value of an a is actually bigger than

[3] Grounding w.r.t. the original $\mathcal{C}$ of P, i.e., not w.r.t. the new $\mathcal{C} = TV_n$.

U_m is covered by the rules in (8) that also introduce any lower values for a value x.

Note that this is a different reduction than the one in [13] from fuzzy dl-programs to dl-programs, where additionally to the closedness restrictions, all combination strategies have to be the ones from Zadeh's logic (i.e., $\otimes = \min$, $\oplus = \max$, and $\ominus = complement$). Our reduction is more general in this sense as it only requires closedness. However, the reduction in [13] allows for disjunctive program, which we do not handle.

We can compute fuzzy answer sets of a fuzzy HEX program by computing the answer sets of the corresponding HEX program:

Proposition 5. *Let P be a disjunction-free fuzzy* HEX *program with closed combination strategies. Then, M is a fuzzy answer set of P iff $M^h = \{\sigma(a,x) \mid M(a) = y, 0 \leq x \leq y\}$ is an answer set of P^h. Vice versa, M^h is an answer set of P^h iff M is a fuzzy answer set of P where M is defined such that $M(a) = \max\{y \mid \sigma_a(y) \in M^h\}$.*

Using the DLVHEX [9] reasoner for reasoning with HEX programs, this proposition thus gives us a method to reason with fuzzy HEX programs as well, in particular by translating them first to HEX programs. Some provisos we have to make in this respect are that the original external functions have to be computable, as well as the external functions associated with the combination strategies. Moreover, the sets of constants, variables, and external predicates under consideration should be finite in order to ensure a finite P^h (note that P itself is by definition finite).

Using the complexity results for HEX programs in [10] and the reduction of HEX programs to fuzzy HEX programs in Proposition 3 we get the following hardness results for different classes of fuzzy HEX programs.

Proposition 6. *Deciding whether a fuzzy* HEX *program without external atoms has a fuzzy answer set is $\mathsf{NEXPTIME}^{\mathsf{NP}}$-hard and $\mathsf{NEXPTIME}$-hard if the program is disjunction-free.*

For programs with external atoms $\#g$, we can deduce the same hardness results if the corresponding function $f_{\#g}$ is decidable in exponential time in $|\mathcal{C}|$.

Proposition 7. *Deciding whether a fuzzy* HEX *program, where for every $\#g \in \mathcal{G}$ the function $f_{\#g}$ is decidable in exponential time in $|\mathcal{C}|$, has a fuzzy answer set is $\mathsf{NEXPTIME}^{\mathsf{NP}}$-hard and $\mathsf{NEXPTIME}$-hard if the program is disjunction-free.*

From the complexity perspective, we even introduce in the absence of external atoms in a fuzzy HEX program P external atoms in the translation P^h to compute the combination strategies as well as the maximum value of an atom. The $\#max$ external function is not introducing extra complexity as the maximum can be calculated in linear time in the size of $gr(P)$. However, the combination strategies can add extra complexity, or lead to undecidability of checking whether there exists a fuzzy answer set in case they are undecidable - note, however, they do not depend on the program at hand. We restrict ourselves thus to *combination-computable* combination strategies:

Definition 5. *A combination strategy $\otimes$ ($\oplus$,$\ominus$) on TV_n is combination-computable if it is closed and its corresponding external function $f_{\#\otimes}$ ($f_{\#\oplus}$, $f_{\#\ominus}$) is decidable in polynomial time. We call a fuzzy HEX program combination-computable if its combination strategies are combination-computable.*

Note that for all the combination strategies we treated in this paper the corresponding functions are decidable in polynomial time, assuming the numbers are encoded in unary format.

Proposition 8. *Deciding whether a combination-computable disjunction-free fuzzy HEX program without external atoms has a fuzzy answer set is in NEXPTIME.*

Proof. The size of the program P^h is linear in the size of $gr(P)$, such that, since the size of $gr(P)$ is in general exponential in the size of P and $\mathcal{C}$, the size of P^h is exponential in the size of P and $\mathcal{C}$. Since we assume that TV_n is fixed, we get that the size of $gr(P^h)$ is polynomial in the size of P^h, and thus, the size of $gr(P^h)$ is exponential in the size of P and $\mathcal{C}$.

Using Proposition 5, checking whether there is a fuzzy answer set of a fuzzy HEX program P, amounts to checking whether there is an answer set of $gr(P^h)$, the latter can be done by a non-deterministic Turing machine in time that is polynomial in the size of $gr(P^h)$ (see, e.g., [10] and [8]). Since the size of $gr(P^h)$ is exponential in the size of P and $\mathcal{C}$, we have that checking whether there is an answer set of $gr(P^h)$ can be done by a non-deterministic Turing machine in time that is exponential in the size of P and $\mathcal{C}$, i.e., in NEXPTIME. □

Using Propositions 6 and 8, we have the following:

Corollary 1. *Deciding whether a combination-computable disjunction-free fuzzy HEX program without external atoms has a fuzzy answer set, is NEXPTIME-complete.*

External atoms in the fuzzy HEX program can introduce complexity, or even undecidability. For fuzzy HEX programs where each external predicate $\#g$ corresponds to a function $f_{\#g}$ that is decidable in a complexity class C in the size of $\mathcal{C}$, we have the following results, again using Proposition 5 and [10]:

Proposition 9. *Deciding whether a combination-computable disjunction-free fuzzy HEX program where each external predicate $\#g$ corresponds to a function $f_{\#g}$ that is decidable in a complexity class C in the size of $\mathcal{C}$ has a fuzzy answer set is in NEXPTIME^C.*

If we restrict ourselves to functions decidable in exponential time, we get, similar as in [10], that the exponential grounding covers for the complexity of the functions:

Proposition 10. *Deciding whether a combination-computable disjunction-free fuzzy HEX program where each external predicate $\#g$ corresponds to a function $f_{\#g}$ that is decidable in exponential time in the size of $\mathcal{C}$ has a fuzzy answer set is in NEXPTIME.*

Using Propositions 7 and 10, we then have the following:

Corollary 2. *Deciding whether a combination-computable disjunction-free fuzzy* HEX *program where each external predicate $\#g$ corresponds to a function $f_{\#g}$ that is decidable in exponential time in the size of $\mathcal{C}$ has a fuzzy answer set is $\mathsf{NEXPTIME}$-complete.*

6 Applications: Service Ranking

In this section, we illustrate the use of fuzzy HEX programs to rank services. We model non-functional properties of services and user preferences as fuzzy HEX programs. For each fuzzy HEX program containing both service description and user preferences represented as rules, the ranking mechanism finds the fuzzy answer sets and their degree of fuzzy match. Based on these degrees, as a final step, the ranked list of corresponding services is constructed.

Assume a user wants to ship a package from Innsbruck to Vienna. The object to be shipped has a value of around 1000 euro according to user's estimation. The weight of the package is 3 Kg and the dimensions are 10/20/10 cm. Furthermore, the user has the following preferences: (1) he wants to pay at most around 70 euro for the service, (2) he wants to pay cash, (3) he wants the package to be ensured in case lost or damage, and (4) he expects the package to be delivered in at most around 36 hours.

Additionally, we have two shipping services *Muller* and *Runner* that potentially could satisfy the user's request. The delivery price for each of the two services depends on the weight, dimension of the package, the distance and delivery time requested by the client. The *Muller* provider presents the following conditions: (1) if the value of the package is at least around 1200 euro and the client payment method is cash the client gets at most 3% discount or free damage insurance, (2) the client has to buy both lost and damage insurances, (3) if the delivery time requested by the user is at least around 40 hours, the user gets a 2% discount from the delivery price. The *Runner* provider presents the following ones: (1) if the value of the package is at least around 1100 euro and the client payment method is cash the client gets at most 4% discount, (2) the client has to buy at least the damage insurance.

The following set of rules represent the background, shared knowledge:

$$\begin{aligned}
\mathit{distance}(\mathit{vienna}, \mathit{innsbruck}, \mathit{485}) &\leftarrow_{\otimes_L} \geq 1\\
\mathit{hasWeight}(\mathit{pack}, \mathit{3}) &\leftarrow_{\otimes_L} \geq 1\\
\mathit{hasDimension}(\mathit{pack}, \mathit{10}, \mathit{20}, \mathit{10}) &\leftarrow_{\otimes_L} \geq 1\\
\mathit{hasValue}(\mathit{pack}) &\leftarrow_{\otimes_L} \mathit{around1000}(\mathit{pack}) \geq \mathit{1}\\
\mathit{hasInsurance}(\mathit{package}, \mathit{lost}, \mathit{5}) &\leftarrow_{\otimes_L} \geq 0.8\\
\mathit{hasInsurance}(\mathit{package}, \mathit{damage}, \mathit{5}) &\leftarrow_{\otimes_L} \geq 0.8\\
\mathit{hasInsurance}(X, \mathit{full}, A) &\leftarrow_{\otimes_L}\\
&\mathit{hasInsurance}(X, \mathit{lost}, B) \otimes_L\\
&\mathit{hasInsurance}(X, \mathit{damage}, C) \otimes_L\\
&\#\mathit{sum}[B, C](A) \geq 1
\end{aligned}$$

$$\begin{aligned}
\mathit{paymentCash} &\leftarrow_{\otimes_L} \geq 1\\
\mathit{paymentCreditcard} &\leftarrow_{\otimes_L} \geq 1\\
\mathit{hasPayment}(X, \mathit{paymentCash})&\\
\oplus_L \mathit{hasPayment}(X, \mathit{paymentCreditcard}) &\leftarrow_{\otimes_L} \geq 1
\end{aligned}$$

where $\mathit{around1000} = \mathit{Tri}(\mathit{900}, \mathit{1000}, \mathit{1100})$, and *Tri* is the triangle function specified in [14]. Note that the rules defining the insurance values in case of lost or damage package have a degree of truth of 0.8. This because, e.g., the exact insurance values are provided by third parties, such as external insurance companies, and service providers have an imprecise knowledge about these values.

The user request and preferences can be encoded as follows:

$$\begin{aligned}
\mathit{query}(X) \leftarrow_{\otimes_L}\ & \mathit{package}(X) \otimes_L \mathit{hasDeliveryPrice}(X, P_D) \otimes_L\\
& \mathit{leqAbout70}(P_D) \otimes_L \mathit{hasInsurance}(X, \mathit{full}, I_F) \otimes_L\\
& \mathit{hasDeliveryTime}(X, T_D) \otimes_L \mathit{leqAbout36}(T_D)\\
& \mathit{hasPayment}(X, \mathit{paymentCash}) \geq \mathit{1}
\end{aligned}$$

In the previous program, we again use a function defined in [14], namely the L-function: $\mathit{leqAbout}36 = L(36, 43)$ and $\mathit{leqAbout}70 = L(70, 75)$ to specify that the expected delivery time to be at most around 36 hours and the expected delivery price to be at most around 70 euro. The predicate *query* collects all packages that fulfill the constraints mentioned above.

The *Muller* service provider restrictions and preferences are encoded as follows:

$$\begin{aligned}
\mathit{discountV}(X, \mathit{3}) \oplus_L \mathit{hasInsurance}(X, \mathit{damage}, \mathit{0}) \leftarrow_{\otimes_L}\ & \mathit{around1200}(X) \otimes_L\\
& \mathit{hasPayment}(X, \mathit{paymentCash}) \geq 1
\end{aligned}$$

$$\begin{aligned}
\mathit{discountT}(X, \mathit{2}) \leftarrow_{\otimes_L}\ & \mathit{not}_{\ominus_L}\ \mathit{leqAbout40}(T_D) \otimes_L\\
& \mathit{hasDeliveryTime}(X, T_D) \geq 1\\
\mathit{totalDiscount}(X, D) \leftarrow_{\otimes_L}\ & \mathit{discountV}(X, B) \otimes_L \mathit{discountT}(X, C) \otimes_L\\
& \#\mathit{sum}[B, C](D) \geq 1\\
\mathit{price}(X, P) \leftarrow_{\otimes_L}\ & \mathit{hasWeight}(X, W) \otimes_L \mathit{hasDimension}(X, D_L, D_W, D_H) \otimes_L\\
& \mathit{distance}(\mathit{Start}, \mathit{End}, \mathit{Dist}) \otimes_L \mathit{hasDeliveryTime}(X, T_D) \otimes_L\\
& \#\mathit{deliveryP}[W, D_L, D_W, D_H, \mathit{Dist}, T_D, f_M](P_D) \otimes_L\\
& \#\mathit{disc}[P_D, D](P_{\mathit{1}}) \otimes_L \mathit{hasInsurance}(X, \mathit{damage}, P_{\mathit{2}}) \otimes_L\\
& \mathit{hasInsurance}(X, \mathit{lost}, P_{\mathit{3}}) \otimes_L \#\mathit{sum}[P_{\mathit{1}}, P_{\mathit{2}}, P_{\mathit{3}}](P) \geq \mathit{1}
\end{aligned}$$

The first rule contains a disjunction in the head used to specify that either a 3% discount for shipping or a free damage insurance is offered. The delivery price computation is done by an external predicate $\#\mathit{deliveryP}[w, \mathit{dim}_l, \mathit{dim}_w, \mathit{dim}_h, \mathit{dis}, \mathit{time}^{\mathit{req}}_{\mathit{del}}, f](P)$, where w is the weight of the package, $[\mathit{dim}_l, \mathit{dim}_w, \mathit{dim}_h]$ is the dimension of the package, *dis* is the distance from source to destination, $\mathit{time}^{\mathit{req}}_{\mathit{del}}$ is the delivery time requested by the client, f is the formula that defines the price computation and P is the computed delivery price for the package. External predicate #*disc* computes a discounted price given an initial price

Algorithm 1: Fuzzy Ranking

```
    Data: Set of services $S_{Ser}$, User request $Q$, Background knowledge $K$,
          represented all as fuzzy HEX programs.
    Result: Order list of services $L_{Ser}$.
    begin
1       $\Omega \longleftarrow \emptyset$, where $\Omega$ is a set of tuples [$service$,$score$] , $\lambda$ - the set of NFPs
        user is interested in;
2       $\beta \longleftarrow \emptyset$, is a set of quadruples [$service$,$nfp$,$nfpvalue$,$degree$];
3       for $s \in S_{Ser}$ do
4           for $nfp \in \lambda$ do
5               if $nfp \in s.nfps$ then
6                   $fuzzy_{prog} = extractNfp(s, nfp) \cup K$;
7                   $[s, nfp, nfpvalue, degree] \Leftarrow evaluate(fuzzy_{prog}, Q)$;
10                  $\beta = \beta \cup [s, nfp, nfpvalue, degree]$;
                end
                else
11                  $\beta = \beta \cup [s, nfp, 0, 1]$;
                end
            end
        end
12      for $s \in \beta$ do
13          $score_s = 0$;
14          for $nfp \in \beta$ do
15              $nfpvalue = \beta.getNFPValue(s, nfp)$;
16              $nfpvalue_{max} = max(\beta.npf)$;
17              $score_s = score_s + degree * \frac{nfpvalue}{nfpvalue_{max}}$;
            end
18          $\Omega = \Omega \cup [s, score_s]$;
        end
19      $L_{Ser} \longleftarrow sort(\Omega)$;
    end
```

and a discount. f_M is the formula used by service *Muller* to define how the delivery price should be computed. $around1200(X)$ is defined similarly as the other *around* predicates, i.e., $around1200(X) = Tri(1000, 1200, 1300)$.

Note that the used combination strategies used so far are Lukasiewizc strategies. However, in our example, one could have used different combination strategies, yielding different results though. The *Runner* service conditions can be encoded similarly as the *Muller* descriptions.

We assume that prior to the service ranking process a discovery process is performed. The discovery process identifies relevant services given a user request by considering semantic descriptions of functional and non-functional aspects of both services and requests. The actual ranking process is presented in Algorithm 1.

First, a fuzzy HEX program containing the background knowledge and a service non-functional property description is created for each service and each of its non-functional properties requested by the user (line 6). In the next step (line 7), the query representing user preferences is evaluated given each program created before. The atoms representing non-functional properties of service are grounded as a result of the previous step and a degree of truth is associated with each of them. Quadruples of form $[service, nfp, nfpvalue, degree]$ are generated. If the non-functional property is not present in the service description the generated quadruple is of form $[service, nfp, 0, 1]$ - the degree of truth is 1 since we know for sure that the value of the NFP is 0 for the given service. The final part of the algorithm (line 15 - line 17) computes an aggregated score for each services, performing first a normalization of the NFPs values and incorporating the degree of truth of every ground atom (line 7). The results are collected in a set of tuples, where each tuple contains the *service id* and the *service score* (line 18). Finally, service scores are sorted and the final ranked list of services is returned (line 19).

The problems of service ranking and selection has been addressed in numerous approaches. Many of them have pointed out the need of fuzzy logic in modeling service descriptions and user preferences. For example, in [16] a fuzzy description logic approach is proposed for automating matching in e-marketplaces. In [19] multiple Quality of Service (QoS) values of services are evaluated and a fuzzy multi-attribute decision making algorithm is proposed to select the best services. The approach does not provide a flexible enough mechanism to model user preferences and services as proposed in our current work. In [20] fuzzy logic is used to evaluate the degree of matching between QoS provided by services and requested by clients. However, the approach is UDDI-based lacking sufficient expressivity for declarative reasoning with user preferences. Furthermore, none of the approaches mentioned before provides support for integration of external data sources or libraries, which is often required in real world settings.

7 Directions for Further Research

As future work, we plan to develop a reasoner for fuzzy HEX programs based on the DLVHEX reasoner, using the translation of fuzzy HEX programs to HEX programs presented in this paper. The implementation of the service ranking algorithm presented in Section 6 together with the evaluation of the approach is also left for the future.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge Press (2003)
3. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference (2004)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, 34–43 (May 2001)

5. de Bruijn, J., Eiter, T., Polleres, A., Tompits, H.: On representational issues about combinations of classical theories with nonmonotonic rules. In: Lang, J., Lin, F., Wang, J. (eds.) KSEM 2006. LNCS (LNAI), vol. 4092, pp. 1–22. Springer, Heidelberg (2006)
6. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The web service modeling language: An overview. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 590–604. Springer, Heidelberg (2006)
7. Chung, L.: Non-Functional Requirements for Information Systems Design. In: Andersen, R., Solvberg, A., Bubenko Jr., J.A. (eds.) CAiSE 1991. LNCS, vol. 498, pp. 5–30. Springer, Heidelberg (1991)
8. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Computing Surveys (CSUR) 33(3), 374–425 (2001)
9. Eiter, T., Ianni, G., Krennwallner, T., Schindlauer, R., Tompits, H.: dlvhex, http://con.fusion.at/dlvhex/
10. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: Proc. of IJCAI 2005 (2005)
11. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining Answer Set Programming with DLs for the Semantic Web. In: Proc. of KR 2004, pp. 141–151 (2004)
12. Lukasiewicz, T.: Fuzzy description logic programs under the answer set semantics for the semantic web. In: RULEML 2006, pp. 89–96. IEEE Computer Society Press, Los Alamitos (2006)
13. Lukasiewicz, T., Straccia, U.: Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. Technical Report 1843-0703
14. Lukasiewicz, T., Straccia, U.: Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 289–298. Springer, Heidelberg (2007)
15. Preist, C.: A conceptual architecture for semantic web services. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298. Springer, Heidelberg (2004)
16. Ragone, A., Straccia, U., Bobillo, F., Di Noia, T., Di Sciascio, E., Donini, F.M.: Fuzzy description logics for bilateral matchmaking in e-marketplaces. Description Logics (2008)
17. Straccia, U.: Fuzzy Logic and the Semantic Web, ch. 4
18. Toma, I., Roman, D., Fensel, D., Sapkota, B., Gomez, J.M.: A multi-criteria service ranking approach based on non-functional properties rules evaluation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 435–441. Springer, Heidelberg (2007)
19. Tong, H., Zhang, S.: A fuzzy multi-attribute decision making algorithm for web services selection based on qos. In: Proc. of APSCC 2006, pp. 51–57. IEEE Computer Society Press, Los Alamitos (2006)
20. Wang, H.-C., Lee, C.-S., Ho, T.-H.: Combining subjective and objective qos factors for personalized web service selection. In: Expert Systems with Applications, pp. 571–584. Elsevier, Amsterdam (2007)

Markup and Component Interoperability for Active Rules

Oliver Fritzen, Wolfgang May, and Franz Schenk

Institut für Informatik, Universität Göttingen
{fritzen,may,schenk}@informatik.uni-goettingen.de

Abstract. We present –on the base of previous papers– a framework for markup, interchange, execution, and interoperability of Active Rules, namely, *Event-Condition-Action (ECA) Rules* over semantically different sublanguages for expressing events, conditions, and actions. The contribution of the present paper is the extension of the MARS meta model of component languages to a meta model of services and an informational infrastructure that is required for a most general framework for specifying and executing active rules over heterogeneous languages. The approach is implemented in the MARS prototype.

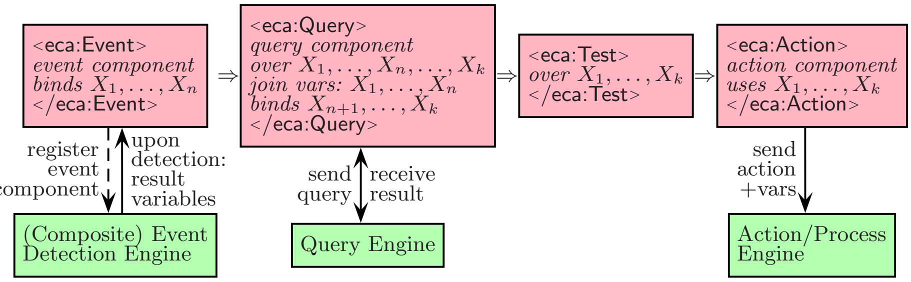
1 Introduction

Rule Markup and Rule Interchange is a highly relevant topic. While a systematic and comprehensive framework for markup of derivation rules, production rules, etc. is developing, no clear direction came up until now for active rules. The main reason for this shortcoming is that the requirements on the markup of active rules, where we consider the most prominent model of *Event-Condition-Action (ECA) Rules*, are significantly more demanding: For the general case, *composite events* and *composite actions*, or even *processes* have to be considered. For both of them, there is a wide variety of specification formalisms. MARS (Modular Active Rules for the Semantic Web) aims at handling this matter by following a modular approach for rule markup, considering the markup and ontology on the generic rule level separately from the markup and ontologies of the components, *and* also *covers* this heterogeneity and openness by a meta-level ontology.

We start the presentation with a condensed overview of MARS ECA rules. We then present the ontology of language classes, service types, and tasks that are involved in a most generic approach for active rules and underlies the operational solution. Finally, we give an overview of the languages implemented in the demonstrator prototype.

Overview – ECA Rules in MARS. For reason of space, we omit details about the semantics of ECA Rules in MARS completely (these can be found in [6, 3, 1]): it is sufficient to know that a set of tuples of variable bindings, similar to the semantics of Datalog Rules, is initialized by the event detection, extended and restricted by queries and tests, and propagated to the action component as illustrated in Figure 1.

$$\begin{aligned}&action(X_1,\ldots,X_n,\ldots,X_k) \leftarrow\\ &\quad event(X_1,\ldots,X_n),\ query(X_1,\ldots,X_n,\ldots,X_k),\ test(X_1,\ldots,X_n,\ldots,X_k)\ .\end{aligned}$$



<eca:Event> event component binds $X_1,\ldots,X_n$ </eca:Event>
<eca:Query> query component over $X_1,\ldots,X_n,\ldots,X_k$ join vars: $X_1,\ldots,X_n$ binds $X_{n+1},\ldots,X_k$ </eca:Query>
<eca:Test> over $X_1,\ldots,X_k$ </eca:Test>
<eca:Action> action component uses $X_1,\ldots,X_k$ </eca:Action>
register event component
upon detection: result variables
send query
receive result
send action +vars
(Composite) Event Detection Engine
Query Engine
Action/Process Engine

Fig. 1. Use of Variables in an ECA Rule

The focus of the present paper is on the *types* of formalisms in which rules and their subexpressions/components are described, and on a *generic* ontology and operational handling of *heterogeneous* languages. A more detailed and formal description of the following can be found in [6, 3, 1].

Figure 2 shows the core structure of ECA rules in MARS and the corresponding types of languages. While the semantics of the ECA rules provides the global semantics, the components are expressed in appropriate languages, that are handled by specific services.

Event Component. For the specification of events, *event algebras* are commonly used [8]. Two levels of specifications are combined (cf. Fig. 3): The specification of the (algebraic) structure of the composite event is given as a *temporal combination* of specifications of the *contributing* atomic events.

Action Component. The action component specifies the actual reaction to be taken. For that, process definition languages over atomic actions are commonly

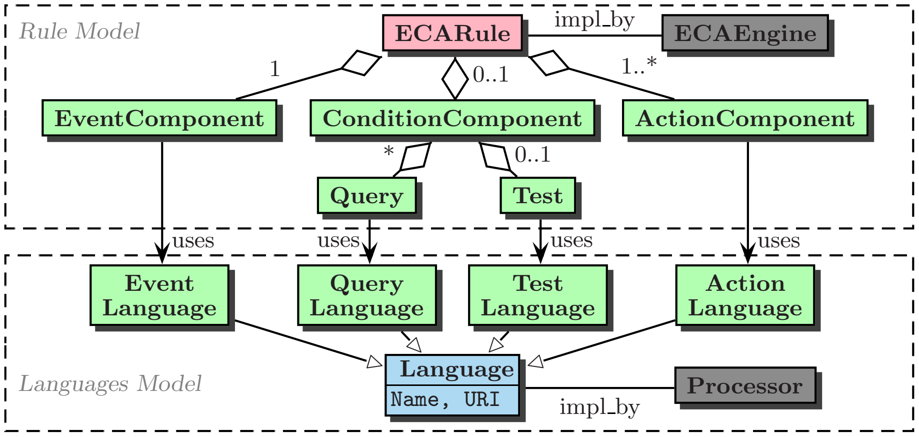


Fig. 2. ECA Rule Components and Corresponding Languages (from [6])

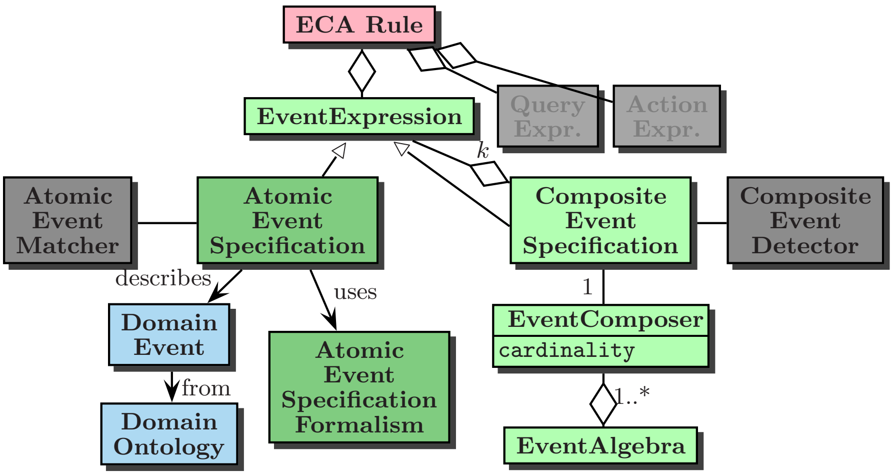
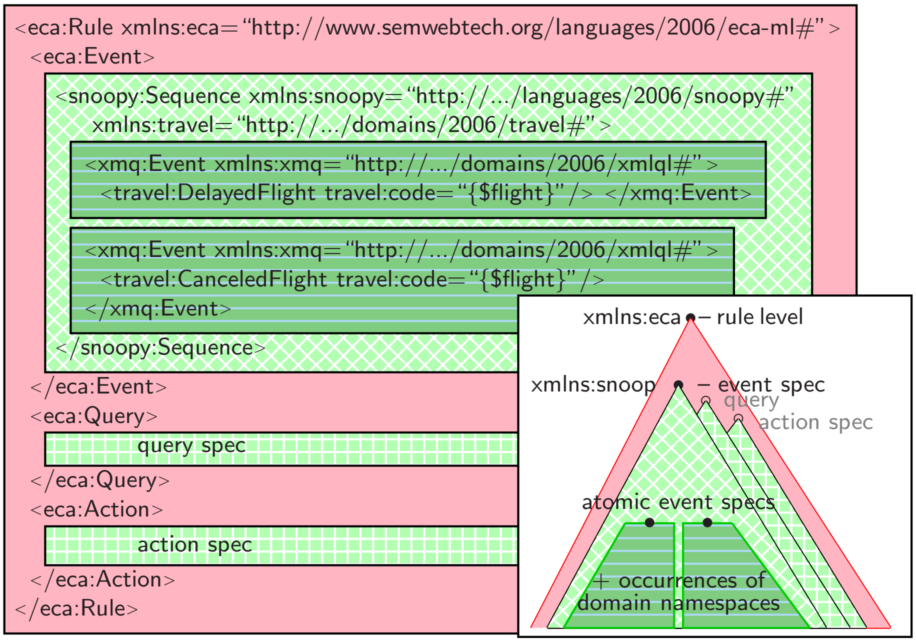


Fig. 3. Event Component: Languages (from [3])

used. The specification of a process, which e.g. includes branching or waiting for a response, can also require the specification of queries to be executed, and of events to be waited for. For that, MARS allows waiting for specified events and evaluation of queries and conditions as regular, executable components of a process; again represented by nested XML expressions (cf. [3]).



```
<eca:Rule xmlns:eca="http://www.semwebtech.org/languages/2006/eca-ml#">
  <eca:Event>
    <snoopy:Sequence xmlns:snoopy="http://.../languages/2006/snoopy#"
        xmlns:travel="http://.../domains/2006/travel#">
      <xmq:Event xmlns:xmq="http://.../domains/2006/xmlql#">
        <travel:DelayedFlight travel:code="{$flight}"/> </xmq:Event>
      <xmq:Event xmlns:xmq="http://.../domains/2006/xmlql#">
        <travel:CanceledFlight travel:code="{$flight}"/>
      </xmq:Event>
    </snoopy:Sequence>
  </eca:Event>
  <eca:Query>
          query spec
  </eca:Query>
  <eca:Action>
          action spec
  </eca:Action>
</eca:Rule>
```

Fig. 4. Nesting of Language Subtrees

Language Borders. Rules and (sub)expressions are represented by XML trees. The conceptual borders are manifested in language borders between the ECA level language and the nested components' languages. The E, C and A components of the rule are subtrees in different languages, corresponding to XML namespaces, as illustrated in Figure 4 (whose details are discussed below in Example 1). Analogous conceptual borders are found between *composite expressions* (by e.g., event algebras) and *atomic subexpressions.*

Example 1 (Nested Languages: Event Component). The sample rule in Fig. 4 illustrates the nesting: the rule reacts on a composite event (specified in the SNOOP [4] event algebra as a sequence) "if a flight is first delayed and later cancelled", and binds the flight number. Two XML-QL-style [5] match expressions contribute the atomic event specifications.

2 The Meta-level: Languages, Services, Tasks

There are languages of different *types* (rules, events, queries/tests, actions), and granularity levels (composite, atomic). Every rule instance uses the ECA-ML rule language and some languages of the other types in a semantically appropriate embedding structure. MARS aims at allowing to embed *arbitrary* such languages of appropriate types by only minimal restrictions on the languages, the services that implement them, and with minimal administrative overhead:

– the information flow between the ECA engine and the event, query, test, and action components is provided by (i) XML language fragments, and (ii) current *variable bindings* (cf. Fig. 1),
– a comprehensive ontology of language types, service types and tasks,
– an open, service-oriented architecture, and
– a *Language and Service Registry (LSR)* that holds information about actual services and how to do the actual communication with them.

The XML *namespaces* are used to identify the language of an XML fragment: namespaces are the languages' URIs. The concrete languages are related to actual services, and the namespace information of a fragment to be processed is used to select and address an appropriate processor. All necessary information what to do with an embedded fragment of a "foreign language" is contained in (i) the language fragment (via the namespace of its root element), (ii) the local knowledge of the currently processing service (i.e., what it expects the fragment to be, and what it wants to do with it), and (iii) the LSR information.

2.1 Languages Types, Service Types, and Tasks

For every type of language there is a specific type of services that provides a specific set of tasks, independent from the concrete language.

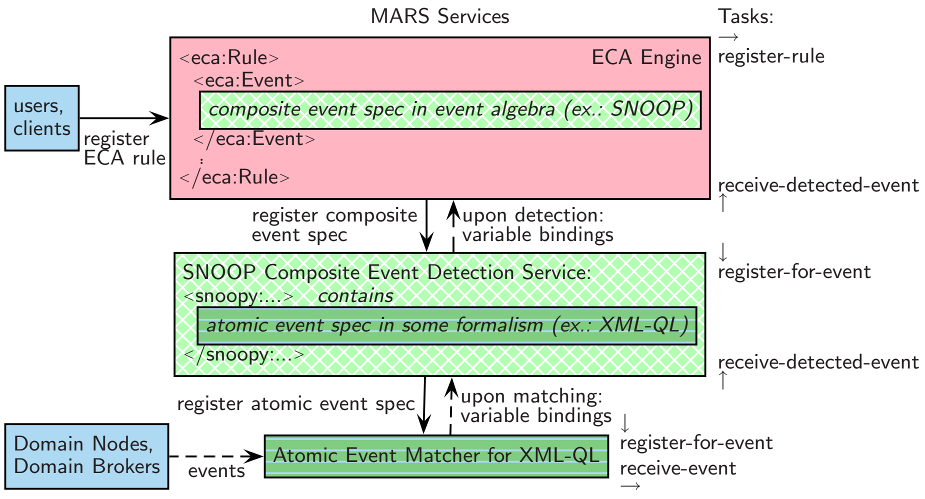

MARS Services
Tasks:
→ register-rule
<eca:Rule>
ECA Engine
<eca:Event>
composite event spec in event algebra (ex.: SNOOP)
</eca:Event>
</eca:Rule>
users, clients
register ECA rule
receive-detected-event ↑
register composite event spec
upon detection: variable bindings
↓ register-for-event
SNOOP Composite Event Detection Service:
<snoopy:...> contains
atomic event spec in some formalism (ex.: XML-QL)
</snoopy:...>
receive-detected-event ↑
register atomic event spec
upon matching: variable bindings
↓ register-for-event
receive-event →
Domain Nodes, Domain Brokers
events
Atomic Event Matcher for XML-QL

Fig. 5. Processing Event Components and Events

Event Specification Languages (specifications of composite or atomic events): composite event specifications are processed by *Composite Event Detection Engines (CEDs)*; atomic event specifications are processed by *Atomic Event Matchers (AEMs)*. In both cases, event specifications can be registered there. Upon occurrence/detection of the event, the registrant will be notified.

Query/Test Languages are handled by query engines. Queries can be sent there, they are answered (synchronously or asynchronously).

Action Languages: Composite and atomic actions are processed by action services. Action specifications can be submitted there for execution.

Additionally, there are the *Domain Languages*: every domain defines a language that consists of the names of actions, classes/properties/predicates, and events. Domain services carry out the real “businesses”, e.g., airlines or universities, by answering queries, executing (atomic) actions, and emitting (atomic) events. *Domain Brokers* [2] implement a portal functionality for a given domain.

Example 2 (Tasks in Event Processing). The architecture part that is relevant for handling events is shown in Figure 5. When a rule is registered at an ECA engine, the ECA engine registers the event component at a *Composite Event Detection Engine (CED)* that understands the respective language (in the example: SNOOP). The CED registers the embedded Atomic Event Specifications (AESs) at appropriate *Atomic Event Matchers (AEMs)* that implement the *Atomic Event Specification Languages (AESLs)*; (in the example: XML-QL). The AEMs determine the domains of the events specified by the AESs and contact appropriate domain brokers to be informed by them about these events. The domain brokers forward relevant atomic events to the AEMs, the AEMs match them against the specifications and inform the CEDs, and the CEDs process the results and inform the ECA engine when a composite event has been detected.

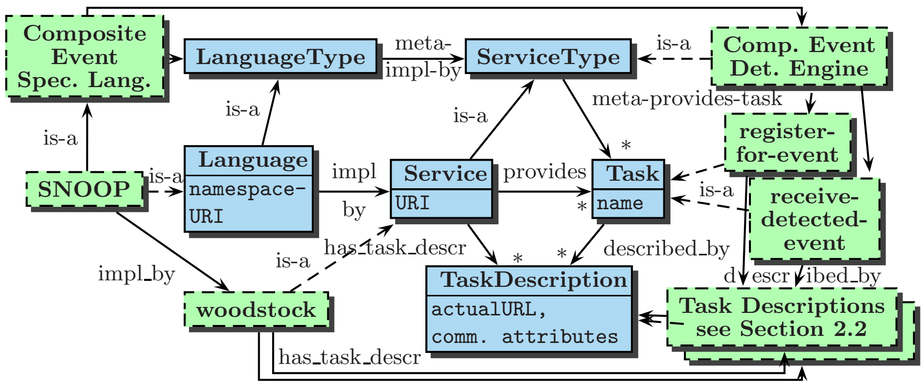
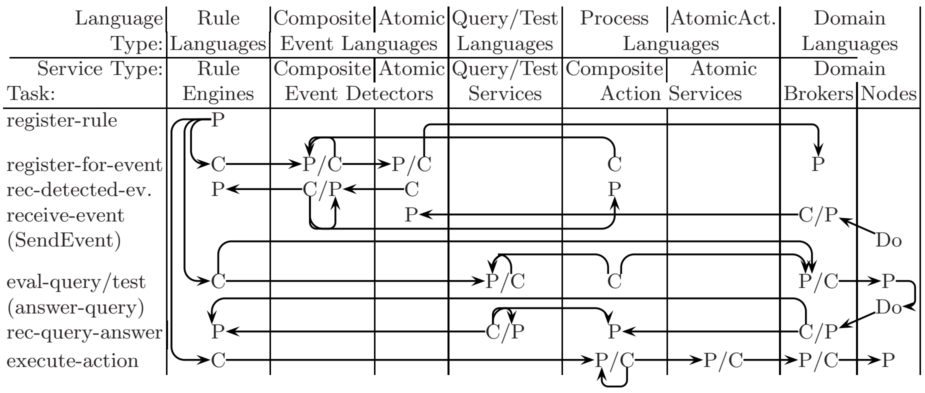


Fig. 6. MARS Ontology of Languages and Services

The MARS *Languages and Services Ontology* is shown in Figure 6; sample instances are denoted by dashed boxes. Figure 7 shows the most important tasks for each service type; additionally, the actual communication flow is indicated.

2.2 Actual Processing

The actual process of determining an appropriate service for handling an embedded component or fragment and organizing the communication is based on the *Language and Service Registry (LSR)* where for each language the concrete information about available services is managed. The communication payload consists of the fragment (including its language identification via the namespace) and the



P: Provides task ; C: Calls task – asynchronous answers will be sent to another task
Arrows from C to P of the same service type represent communication between different services of the same type (e.g., nesting of different event algebras).
Do: does something (inherent behavior of domain nodes).

Fig. 7. Services and Tasks

```
<mars:EventAlgebra rdf:about="http://.../languages/2006/snoopy#" >
  <mars:is-implemented-by>
    <mars:CompositeEventDetectionEngine xml:base="http://.../services/2007/woodstock/"
        rdf:about="http://www.semwebtech.org/services/2007/woodstock">
      <has-task-description> <TaskDescription>
          <describes-task rdf:resource="&mars;/ced#register-event-pattern"/>
          <provided-at rdf:resource="register"/> <input>element register</input>
          <Reply-To>body</Reply-To> <Subject>body</Subject>
          <variables>*</variables>
        </TaskDescription> </has-task-description>
        :
    </mars:CompositeEventDetectionEngine>
  <mars:is-implemented-by>
</mars:EventAlgebra>
```

Fig. 8. MARS LSR: LSR entry with Service Description Fragment for SNOOP

variable bindings; in case that the answer will be returned asynchronously, an identifying subject is also needed. For asynchronous answers, the payload is the answer (in XML markup) and the identifying subject. Operationally, this is performed in a generic way by a *Generic Request Handler (GRH)* (implemented by a Java class that is used by all sample services).

Communication Information about Concrete Tasks. Given an embedded fragment in some language to be processed, the namespace of the fragment is used to obtain a service description of an appropriate service from the LSR. For every concrete task of an actual service, the communication information is given in the *Task Description*, cf. Figure 8:

– the actual URL (as a service supports multiple tasks, each of them may have an own URL, which is not necessarily related to the service's URI),
– whether it supports to submit a set of tuples of variables,
– information about the required message format:
  • send reply-to address and subject in the message header or in the body,
  • whether it requires a certain wrapper element for the message body,
– whether it will answer synchronously or asynchronously.

All MARS ontologies and an LSR snapshot in XML/RDF syntax can be found at `http://www.dbis.informatik.uni-goettingen.de/MARS/#mars-ontologies`.

Example 3. When the above rule is registered at the ECA engine, the event component <snoopy:Sequence> is to be registered at the appropriate CED. The ECA engine asks the LSR for a service that implements SNOOP (using the namespace URI) to get the complete service description of the "woodstock" SNOOP engine as shown in Figure 8. The ECA engine (or more detailed, its GRH) takes the task description for the task ced#register-event-pattern that describes how to format the request, and that it has to be sent to http://www.semwebtech.org/services/2007/woodstock/register. In the task description, the <variables> entry indicates whether a set of tuples can be sent, or if only one tuple at a time can be processed. The communication of the detected events is done analogously via the receive-detected-event task of the ECA engine.

2.3 A Comprehensive Set of Sample Languages

The MARS demonstrator provides several sample languages on the XML and RDF level (cf. the online LSR, and the online demo interface at http://www.semwebtech.org/mars/frontend/): An XML-QL-style [5] pattern-based mechanism for specification of atomic event patterns, queries, and atomic actions, the SNOOP event algebra [4] for Composite Event Specifications, XPath and XQuery for *opaque* queries (i.e., non-markupped CDATA contents), and the *CCS – Calculus of Communicating Systems* process algebra [7] for Composite Actions. The architecture of the application domain level, consisting of domain nodes and domain brokers, is described in [2]. The MARS framework is open for foreign component languages and other sublanguages. Languages that have an XML markup smoothly integrate as shown above.

3 Conclusion

We described an open, XML-based framework for rule markup in a heterogeneous, multi-language setting. The key concept is an ontology of language types, corresponding service types, and language-type-specific tasks. Communication between the respective Web Services is established at runtime based on the information contained in the rule markup and the *Languages and Service Registry*.

References

[1] Alferes, J.J., Amador, R., Behrends, E., Fritzen, O., May, W., Schenk, F.: Pre-Standardization of the Language. Technical Report I5-D10, REWERSE EU FP6 NoE (2008), http://www.rewerse.net

[2] Behrends, E., Fritzen, O., Knabke, T., May, W., Schenk, F.: Rule-Based Active Domain Brokering for the Semantic Web. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 259–268. Springer, Heidelberg (2007)

[3] Behrends, E., Fritzen, O., May, W., Schenk, F.: Embedding Event Algebras and Process Algebras in a Framework for ECA Rules for the Semantic Web. Fundamenta Informaticae 82, 237–263 (2008)

[4] Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.-K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: VLDB (1994)

[5] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: XML-QL: A Query Language for XML. WWW8. W3C (1999), www.w3.org/TR/NOTE-xml-ql

[6] May, W., Alferes, J.J., Amador, R.: Active Rules in the Semantic Web: Dealing with Language Heterogeneity. In: Adi, A., Stoutenburg, S., Tabet, S. (eds.) RuleML 2005. LNCS, vol. 3791, pp. 30–44. Springer, Heidelberg (2005)

[7] Milner, R.: Calculi for Synchrony and Asynchrony. Theoretical Computer Science, 267–310 (1983)

[8] Paton, N.W.: Active Rules in Database Systems. Springer, Heidelberg (1999)

On Reducing Redundancy in Mining Relational Association Rules from the Semantic Web

Joanna Józefowska, Agnieszka Ławrynowicz, and Tomasz Łukaszewski

Institute of Computing Science, Poznan University of Technology,
ul. Piotrowo 2, 60-965 Poznan, Poland
{jjozefowska,alawrynowicz,tlukaszewski}@cs.put.poznan.pl

Abstract. In this paper we discuss how to reduce redundancy in the process and in the results of mining the Semantic Web data. In particular, we argue that the availability of the domain knowledge should not be disregarded during data mining process. As the case study we show how to integrate the semantic redundancy reduction techniques into our approach to mining association rules from the hybrid knowledge bases represented in OWL with rules.

1 Introduction

The methods for mining the Semantic Web data should be able to operate on the representation languages lying in the Semantic Web stack. Hence, they should take into account the relations existing in "linked" data and incorporate, into the data mining process, domain knowledge represented in *Web Ontology Language* (OWL) and rule languages (*Rule Interchange Format* RIF, *Semantic Web Rule Language* SWRL). Due to first order nature of the representation formalisms underlying the languages from the Semantic Web stack (e.g. description logics), the algorithms of data mining in this setting are supposed to follow into *relational data mining* category. Data mining methods typically perform a kind of search through the space of hypotheses. In relational setting, in comparison to simple representations, like "attribute-value" one, the space of hypotheses may be very large. In case of expressive languages it becomes very expensive to generate and test all syntactically valid hypotheses. Moreover, managing all discovered patterns may become hard to the user which contradicts the premier goal of data mining which is to summarize and explain the data.

In fact, it is not neccesserily needed that all syntactically valid hypotheses are generated and discovered. Many of them may cover the same examples, that is may be equivalent. In this paper, we discuss the issues related to semantic redundancy in the context of mining relational frequent patterns and relational association rules from the Semantic Web. The main contribution of this paper is the identification and the incorporation of the semantic redundancy techniques into our proposed framework to mining frequent patterns and association rules from hybrid knowledge bases represented in description logics with $\mathcal{DL}$-safe rules. The framework was originally presented in [3]. The algorithms for mining patterns within the framework were proposed in [4].

2 Preliminaries

Representation of Data and Patterns. We assume mining patterns and rules in a combined knowledge base (KB, P), where KB is description logics component and P is a program containing a set of positive (disjunctive) Datalog rules. The formalism used in our approach is that of $\mathcal{DL}$-safe rules [7], but with the subset of description logics restricted to $\mathcal{SHIF}$. Example (KB, P), further exploited in the discussion of the problems with redundancy and in the discussion of the solutions, is presented below.

Example 1 (Example knowledge base (KB, P)). Given is a knowledge base describing bank services. Non-DL-predicates names start with prefix *p_*. All rules in P are $\mathcal{DL}$-safe, that is applicable only to explicitly introduced individuals.

Terminology in KB	
$Client \equiv \exists\ isOwnerOf$	A client is defined as an owner of something.
$\top \sqsubseteq \forall\ isOwnerOf.(Account \sqcup CreditCard)$	The range of *isOwnerOf* is a disjunction of *Account* and *CreditCard*.
$\exists isOwnerOf^{-} \sqsubseteq Property$	Having an owner means being a property.
$Gold \sqsubseteq CreditCard$	*Gold* is a subclass of *CreditCard*.
$relative \equiv relative^{-}$	The role *relative* is symmetric.
$\top \sqsubseteq \forall\ relative.Person$	The range of *relative* is *Person*.
$Account \sqsubseteq \exists\ isOwnerOf^{-}$	All accounts have an owner.
$Account \equiv \neg\ Person$	*Account* is disjoint with *Person*.
$Account \equiv \neg\ CreditCard$	*Account* is disjoint with *CreditCard*.
$Person \equiv \neg\ CreditCard$	*Person* is disjoint with *CreditCard*.
Assertions in KB	
Person(Anna).	Anna is a person.
isOwnerOf(Anna,account1).	Anna is an owner of account1.
relative(Anna,Marek).	Anna is a relative of Marek.
Person(Jan).	Jan is a person.
isOwnerOf(Jan,creditcard1).	Jan is an owner of creditcard1.
Person(Marek).	Marek is a person.
isOwnerOf(Marek,account1).	Marek is an owner of account1.
Account(account2).	Account2 is an account.
Axioms in P	
p_familyAccount(x,y,z) $\leftarrow$ *Account(x), isOwnerOf(y,x), isOwnerOf(z,x), relative(y,z)*	*p_familyAccount* is an account that is co-owned by at least two relatives.
p_man(x) $\vee$ *p_woman(x)* $\leftarrow$ *Person(x)*	A person is either a man or a woman.
p_sharedAccount(x,y,z) $\leftarrow$ *p_familyAccount(x,y,z)*	Family account is a shared account.

Usually the task of mining association rules is composed of two steps: finding *frequent patterns* and generating *association rules* from the discovered patterns. The patterns in our approach have the form of *conjunctive queries* over combined knowledge base (KB, P). The *answer set* of the query contains individuals of a user-specified reference concept $\hat{C}$. We assume that the queries are *positive*, that is do not contain negative literals. Moreover, we assume that the queries are $\mathcal{DL}$*-safe*, that is all variables in such a query are bound to individuals explicitly occurring in the knowledge base, even if they are not returned as part of the query answer. Let Q_1 and Q_2 be such patterns that $Q_1 \subseteq Q_2$, that is the set of atoms of query Q_1 is the subset of atoms of query Q_2. Relational association rule, *query extension* [2], is an implication of the form $Q_1 \rightarrow Q_2$.

Example 2 (Example patterns and rules). Consider the knowledge base (KB, P) from Example 1. Assuming that $Person$ is the reference concept $\hat{C}$, the following patterns, queries over (KB, P), may be built:

$$Q_{ref}(key) = ?-Person(key)$$

$$Q_1(key) = ?-Person(key), isOwnerOf(key, x)$$

$$Q_2(key) = ?-Person(key), isOwnerOf(key, x), p\_familyAccount(x, key, z)$$

where Q_{ref} is a trivial query, *reference query*, that counts the number of instances of the reference concept. Below is the example association rule:

$$?-Person(key), isOwnerOf(key, x) \rightarrow ?-Person(key), isOwnerOf(key, x), Account(x)$$

The meaning of the rule is that "whenever a person is an owner of something he or she may be possibly an owner of the account".

With regards to our work presented in [4], patterns, that is conjunctive queries, have been extended here from the ones over KB to the ones over (KB, P). What follows, the conjunctive queries, as presented in this paper, can contain intensional predicates from Disjunctive Datalog program P, that is also n-ary predicates.

Task Formulation. The task of frequent pattern discovery is to find all patterns whose support exceeds a minimum support threshold *minsup*, specified by the user, that is to find all *frequent patterns*. The task of frequent and confident association rule discovery is to find all frequent rules whose confidence exceeds a given minimum confidence threshold, *minconf*. The *support* of query Q with respect to the knowledge base (KB, P) is defined as the ratio between the number of instances of reference concept $\hat{C}$ that satisfy query Q and the total number of instances of reference concept $\hat{C}$. The support of association rule $Q_1 \rightarrow Q_2$ is the support of query Q_1. The *confidence* of the association rule $Q_1 \rightarrow Q_2$ w.r.t combined knowledge base (KB, P) is computed as the ratio of the support of Q_2 to the support of Q_1.

3 Semantic Redundancy in Mining Frequent Relational Patterns

In this section we will focus on mining frequent patterns. As association rules are postprocessed from the patterns, the semantic redundancies in rules may be directly inherited from those generated while mining patterns. Let us identify the points where one can search for the reasons for semantic redundancy while mining frequent patterns, which are as follows: (1) representation language, used by mining algorithm, unable to catch the relationships holding in a knowledge base; (2) generality measure between patterns unable to catch the redundancy in a single pattern as well as in a set of patterns; (3) data mining algorithm that does not make use of the semantic relationships holding in a knowledge base and/or does not efficiently/fully employ the generality measure. Let us discuss these points in the context of frequent pattern mining from the Semantic Web.

Representation of Domain Knowledge. Originally, most of the methods for mining frequent relational patterns have been designed to operate on knowledge bases represented as Logic Programs (generally in Datalog variant) or just used first order logic notation. Examples are WARMR [2], FARMER [8] and c-armr [9]. Thus, by definition, they are not able to use domain knowledge of the form of description logics axioms that could not be rewritten into Datalog. Datalog rules require all variables to be universally quantified and thus it is impossible to assert the existence of unknown individuals. For example, it is impossible to assert that all accounts must have an owner. Let us show the example how the inability to handle all the relationships represented in a knowledge base may affect redundancy. Consider the following pattern:

$$Q(x) = ? - Account(x), Property(x)$$

The second literal of Q is semantically redundant, as from the knowledge base we already know that every account is a property. In the knowledge base it is stated that if something has an owner, than it is a property. Moreover, it is stated that every account has an owner. For example, $account2$ is a property, even if there is nowhere written so, and the owner of $account2$ is nowhere specified. In Datalog such deduction, in order to catch the redundancy, is impossible.

SPADA [6] (its further version is named $\mathcal{AL}$-QuIn [5]) has been the only approach so far that has aimed at frequent pattern discovery in combined knowledge base, consisting of Datalog and description logics. SPADA/$\mathcal{AL}$-QuIn uses $\mathcal{AL}$-log, which is the combination of Datalog with $\mathcal{ALC}$ description logics language. Patterns in SPADA/$\mathcal{AL}$-QuIn are represented as constrained Datalog clauses.

Generality Measure. *Generality measure* between patterns is used in frequent pattern mining systems to build more specific patterns from more general ones. Most of the relational data mining methods, e.g. WARMR and FARMER, during generation of patterns use θ-*subsumption* as the generality measure. θ-*subsumption* is a syntactic generality relation and as such is not strong enough to capture semantic redundancies. Consider the knowledge base from Example 1. For such setting WARMR discovers queries like the one presented below:

$$Q(y) = ? - p\_woman(y), p\_familyAccount(x, y, z), p\_sharedAccount(x, y, z)$$

In the domain knowledge $p\_familyAccount$ is defined as a type of p_shared $Account$. This makes second literal of Q, $p\_sharedAccount(x, y, z)$, semantically redundant. Query Q is the example of a redundancy in a single pattern. Using syntactic generality measure generates redundancy effects in a set of patterns, too. Consider, for example, the following, semantically equivalent queries that would be both discovered in WARMR:

$$Q_0(y) = ? - p\_woman(y), p\_familyAccount(x, y, z)$$

$$Q(y) = ? - p\_woman(y), p\_familyAccount(x, y, z), p\_sharedAccount(x, y, z)$$

In c-armr, semantic generality measure is used, that have similar effect as generalized subsumption [1]. It allows to avoid the abovementioned redundancies as well as in a single pattern as in a set of patterns. SPADA has been conceived to use semantic generality measure (query containment), too, but it does not fully use it in an algorithm for pattern mining what is discussed in next paragraph.

Algorithms. Relational frequent pattern mining algorithms usually generate patterns according to a specification in a *declarative bias*. Declarative bias allows to specify the set of atom templates describing the atoms to be used in queries. Common solution is to take one atom template after the other to build the refinements of a query, in order in which the templates are stored in a declarative bias directives. Such solution is adopted in WARMR, FARMER, c-armr. However, the solution does not make use of a semantic relationships between predicates in atoms, causing redundant computations. Consider the patterns:

$$Q_1(y) = ? - p\_woman(y), p\_sharedAccount(x, y, z)$$

$$Q_2(y) = ? - p\_woman(y), p\_familyAccount(x, y, z)$$

Assume that pattern Q_1 has been found infrequent. Thus, generating pattern Q_2 is useless, since $p\_familyAccount$ is more specific than $p\_sharedAccount$. C-armr would generate and test both queries anyway. If some taxonomic information would be used to systematically generate refinements, the redundant computation could be avoided. In SPADA, taxonomic information is used only with regards to concept hierarchies. That is, patterns are refined by replacing more general concepts by more specific ones in the constraints of the constrained Datalog clause. Any technique using taxonomic information is not reported with regards to the Datalog predicates. It means, the same scheme of refining patterns, described above, is applied in SPADA/$\mathcal{AL}$-QuIn, too.

Another point, related to the above one, is that SPADA, despite of being conceived to use semantic generality measure, does not apply it to prune semantically redundant patterns. It is not used either to prune patterns semantically redundant, due to redundant literals nor to prune semantically equivalent patterns. That is, similar pattern as described in last paragraph w.r.t. WARMR, would be generated by SPADA:

$$q(y) \leftarrow p\_woman(y), p\_familyAccount(x, y, z), p\_sharedAccount(x, y, z) \ \& \ Client(y)$$

Also, there is no solution in SPADA algorithm to check the redundancy using the knowledge linking Datalog and description logics component (like the rule defining $p\_familyAccount$). The following clause may be generated:

$$q(x) \leftarrow p\_familyAccount(x, y, z), p\_woman(y) \ \& \ Account(x), Client(y)$$

while from the (KB, P) already follows the constraint $Client$ on variable y.

4 Dealing with Redundancy in Mining Frequent Patterns from the Semantic Web

Representation of Domain Knowledge. $\mathcal{DL}$-safe rules combine description logics with (Disjunctive) Datalog. Hence, this formalism does not introduce any restrictions described in previous section. Recall, that WARMR, FARMER and c-armr operate only on Datalog. SPADA/$\mathcal{AL}$-QuIn uses $\mathcal{AL}$-log, what restricts its patterns to contain only DL concepts, but any roles. In turn, both, concepts and roles, are allowed in our patterns. In our (KB, P), DL-atoms may occur in the body and in rule heads as well, which is not possible in $\mathcal{AL}$-log, which allows only for concepts to be used as constraints in the body of a clause. The internal representation language used in SPADA/$\mathcal{AL}$-QuIn corresponds to Datalog, while that used in our method to, more expressive, Disjunctive Datalog.

Testing Query Satisfiability. In our approach, before submitting a query to test its frequency, we employ a knowledge base to perform *semantic* tests. First test consists of determining query satisfiability, further ones test for semantic redundancy in a single pattern or in a set of patterns.

The test for checking satisfiability of query Q with regards to knowledge base (KB, P) consists of checking whether $(KB, P) \cup \exists \mathbf{x}, \mathbf{y} : Q$ is satisfiable, that is, whether there is a model of (KB, P) in which there is some valuation for distinguished variables $\mathbf{x}$ and nondistinguished variables $\mathbf{y}$. In practice, the satisfiability test is performed as defined below.

Definition 1. *Query $Q(\mathbf{x}, \mathbf{y}) =? - B_1, ..., B_n$, where B_i denote atoms of the query, is* satisfiable *w.r.t. combined knowledge base (KB, P) iff (KB, P) $\cup\, B_1\theta, ..., B_n\theta$ is satisfiable, where θ is a Skolem substitution.*

For example, we know apriori that it is useless to submit the following query as it cannot have any answer due to its unsatisfiability:

$$Q(x) =? - Person(x), Account(x)$$

Semantic Generality Measure. The starting point for applying the techniques for semantic redundancy reduction is choosing semantically-aware generality measure. We define a *generality measure* between two patterns as *query containment* (or *subsumption*) relation.

Definition 2 (Generality Relation). *Given two queries Q_1 and Q_2 we say that query Q_1 is at least as general as query Q_2 under query containment, $Q_1 \succeq Q_2$, iff query Q_2 is contained in query Q_1.*

Since the generality relation between two patterns is defined in terms of query containment it relies on query answering. The query answering procedure [7] takes into account background theory in a form of a combined knowledge base (KB, P), with respect to which the queries are to be evaluated.

In practice, in our approach, the test of generality between two patterns (queries) relies on the assumption that the queries are $\mathcal{DL}$-safe. $\mathcal{DL}$-safe queries are queries without truly undistinguished variables. That is, even though one can specify the variables whose bindings are supposed to be returned in the answer set, the bindings of the remaining variables (undistinguished ones) are also computed. Taking into account the abovementioned feature, query containment can be tested as follows. Let's assume that it is tested whether $Q_1(\mathbf{x}, \mathbf{y_1})$ is contained in $Q_2(\mathbf{x}, \mathbf{y_2})$, where $\mathbf{x}$ and $\mathbf{y_i}$ denote distinguished and undistinguished variables respectively. Then it is asserted $Q_1(\mathbf{a}, \mathbf{b})$ where $\mathbf{a}$ and $\mathbf{b}$ are new individuals, and tested whether $\mathbf{a}$ is the answer to query Q_2. For example, query Q_2 from Example 2 is contained in query Q_1, hence Q_1 is more general than Q_2 ($Q_1 \succeq Q_2$). The inverse does not hold.

Semantic generality measure serves to perform tests for semantic redundancy before submitting the query. Similarly as proposed in c-armr, we test generated queries to obtain only those that are not *semantically redundant*. Firstly, we check whether a query has redundant literals, that is the ones that can be deduced from the other literals in a query. Secondly, we check whether there are frequent queries already found in earlier steps that are *semantically equivalent* to a newly generated candidate. To avoid the former kind of redundancy, generated queries are tested for *semantic freeness*. Only those of them that are *semantically free* are kept for further processing. The notion of semantic freeness has been introduced in [9]. It is adapted to our setting as follows.

Definition 3 (Semantically Free Pattern). *A pattern Q is* semantically free *or* s-free *w.r.t combined knowledge base (KB, P) if there is no pattern Q', with any literal(s) from Q removed, such that $Q \succeq Q'$ (Q is more general than Q').*

Let us consider the following queries to the knowledge base from Example 1:

$$Q_1(key) =? - Account(key), isOwnerOf(x, key)$$

$$Q_2(key) =? - Account(key), isOwnerOf(x, key), Client(x)$$

Query Q_1 is s-free. Query Q_2 is not because of literal $Client(x)$ that can be deduced from other literals of Q_2. More specifically, $Client(x)$ can be deduced from literal $isOwnerOf(x, key)$ as from the axioms in the knowledge base follows that any object being asserted to domain of $isOwnerOf$ is a $Client$.

Testing whether a query is semantically equivalent to already found frequent one is performed by a search on a set of already found, frequent patterns.

Using Taxonomies. In our method, we propose to use concept and role taxonomies from KB to systematically build pattern refinements. The refinements are not generated based on a flat list of atom templates, but based on tree-shaped taxonomies of predicates. If a "superclass" refinement of some predicate is found infrequent, its "subclass" is not tested. Consider the following queries:

$$Q_1(key) =? - Person(key), isOwnerOf(key, x), CreditCard(x)$$

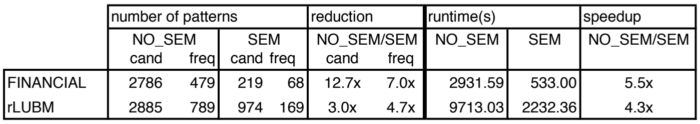
$$Q_2(key) =? - Person(key), isOwnerOf(key, x), Gold(x)$$

If query Q_1 is found infrequent, then, its specialization Q_2, is not generated at all, due to subclass relation between $Gold$ and $CreditCard$.

Implementation. Figure 1 presents chosen experimental results (no semantics from (KB, P) used during candidate patterns generation, *NO_SEM*, w.r.t. semantic techniques described in this section applied, *SEM*). The patterns were built from all predicates with any extension. The maxiumum length of patterns was 4. The reference concepts for FINANCIAL[1] and rLUBM[2] datasets were respectively

[1] http://www.cs.put.poznan.pl/alawrynowicz/financial.owl

[2] LUBM benchmark, number of universities set to 1, with rules from http://pellet.owldl.com/papers/kolovski06extending.pdf



	number of patterns				reduction		runtime(s)		speedup
	NO_SEM		SEM		NO_SEM/SEM		NO_SEM	SEM	NO_SEM/SEM
	cand	freq	cand	freq	cand	freq			
FINANCIAL	2786	479	219	68	12.7x	7.0x	2931.59	533.00	5.5x
rLUBM	2885	789	974	169	3.0x	4.7x	9713.03	2232.36	4.3x

Fig. 1. Experimental results

Client and *Person*, the *minsup* thresholds 0.2 and 0.3.

The release of our method with semantic redundancy reduction features, SEMINTEC-ARM (SEMINTEC-*Association Rule Miner*), is available as the source code in Java[3], together with the results of the experimental evaluation. As an underlying reasoning engine, KAON2 [4] is used.

Summary. Finally, in Table 1, we provide the comparison of the semantics-related features of the approaches to relational frequent pattern mining.

Table 1. Semantic features of the approaches to relational frequent pattern mining

	Knowledge representation			Method			
	$\mathcal{DL}$ component	Datalog component	disjunctive rules	semantic generality measure	semantically free (non-redundant) patterns	only one pattern from each equivalence class	taxonomies directly used in refining patterns
WARMR		x					
FARMER		x					
c-armr		x		x	x	x	
SPADA/$\mathcal{AL}$-QuIn	x	x		x			x
SEMINTEC-ARM	x	x	x	x	x	x	x

5 Conclusions

We have discussed the semantic redundancy reduction techniques in the context of mining relational frequent patterns and association rules from the Semantic Web. The experimental evaluation shows that application of the techniques allows to achieve less, more compact results in a shorter time.

References

1. Buntine, W.: Generalized subsumption and its applications to induction and redundancy. Artificial Intelligence 36(2), 149–176 (1988)
2. Dehaspe, L., Toivonen, H.: Discovery of frequent Datalog patterns. Data Mining and Knowledge Discovery 3(1), 7–36 (1999)

[3] http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

[4] http://kaon2.semanticweb.org/

3. Józefowska, J., Ławrynowicz, A., Łukaszewski, T.: Towards discovery of frequent patterns in description logics with rules. In: Adi, A., Stoutenburg, S., Tabet, S. (eds.) RuleML 2005. LNCS, vol. 3791, pp. 84–97. Springer, Heidelberg (2005)
4. Józefowska, J., Ławrynowicz, A., Łukaszewski, T.: Frequent pattern discovery in OWL DLP knowledge bases. In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS (LNAI), vol. 4248, pp. 287–302. Springer, Heidelberg (2006)
5. Lisi, F.A., Esposito, F.: Efficient Evaluation of Candidate Hypotheses in $\mathcal{AL}$-log. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 216–233. Springer, Heidelberg (2004)
6. Lisi, F.A., Malerba, D.: Inducing Multi-Level Association Rules from Multiple Relation. Machine Learning Journal 55, 175–210 (2004)
7. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 549–563. Springer, Heidelberg (2004)
8. Nijssen, S., Kok, J.N.: Faster Association Rules for Multiple Relations. In: Proceedings of the IJCAI 2001, pp. 891–897 (2001)
9. De Raedt, L., Ramon, J.: Condensed representations for Inductive Logic Programming. In: Proc. of the KR 2004, pp. 438–446 (2004)

A Triple-Oriented Approach for Integrating Higher-Order Rules and External Contexts

Andreas Billig[1,2]

[1] Jönköping University, Jönköping, Sweden
Andreas.Billig@jth.hj.se
[2] Fraunhofer ISST, Berlin, Germany
Andreas.Billig@isst.fraunhofer.de

Abstract. In recent years, many researchers in the area of reasoning have focussed on the adoption of rule languages for the Semantic Web that led to remarkable approaches offering various functionality. On one hand, this included language elements of the rule part itself like contexts, higher-orderness, and non-monotonic negation. On the other hand, the proper integration with ontology languages like RDF and OWL had to consider language-specific properties like disjunctivity as well as the demand for using existing external components. The paper proposes a Triple-oriented hybrid language that integrates the mentioned language elements of both aspects following the expressiveness of locally stratified datalog. It introduces fixpoint semantics as well as pragmatic extensions for defining transformations between fact bases. A partial implementation is based on stratified, semi-naive evaluation, and static filtering.

1 Introduction

Many mature approaches from the area of rules languages and reasoning have been utilized for the purposes of the semantic web. Core elements like higher-orderness, disjunction and non-monotonic negation have been investigated and combined. Examples for higher-order languages in our scope are HiLog, F-Logic, and HEX [1][2][3]. Disjunction plays the crucial role in dealing with ontology languages like OWL as a web-customized standardization of Description Logics (DL) [4]. Combining DL and rule languages without constraining its possibilities to define arbitrary predicates are reflected in recent approaches like KAON2, DL+Log, HEX, and Datalog^{DL} [5][6][3][7]. Whereas the first three ones are based on disjunctive datalog and support homogenous rule integration the last one is based on Horn Logic and supports hybrid integration. Non-monotonic negation is provided by DL+Log and HEX. Moreover, HEX introduces a concept for integrating existing external components like OWL reasoners or RDF bases. The language Triple [8] introduces contexts for semantic web applications in the spirit of early roots [9]. Especially the concept of parameterized contexts enables an elegant way to express transformations between information representable by fact sets and rules.

As argued in the corresponding research of existing approaches every mentioned language element is highly useful for semantic web applications based on rules and DL. We want to propose a hybrid approach that pragmatically comprises not only a subset but all language elements, namely contexts (in the sense of [8]), higher-orderness, non-monotonic negation, and disjunctivity. One essential constraint for us was to follow the expressiveness of (locally) stratified datalog which enables us to reuse widely-used techniques for an efficient reasoner. The organization of the paper is as follows: section 2 presents syntax and examples. Section 3 defines the semantics of our language by a fixpoint procedure. Some pragmatic aspects concerning transformations are presented in section 4.

2 Syntax

Basic Elements. The alphabet of the language consists of disjoint sets of *constants* $\mathbf{C}$,*variables* $\mathbf{V}$, and the usual punctation marks. Elements of $\mathbf{C}$ can have the form of RDF resources, i. e., a namespace can be prefixed. If $v \in \mathbf{V}, c_1, \ldots, c_n \in \mathbf{C}$ $(n \geq 1)$ then the expression v<$c_1, \ldots, c_n$> is called *filtered variable.* A *term* is a constant or a (filtered) variable. If $t_0, \ldots, t_n (n \geq 0)$ are terms the tuple form $(t_0 \ \ldots \ t_n)$ is called *filtered tuple* (short *f-tuple*). Let a, b be f-tuples. Then a @ b is called *(contextual higher-order) atom*[1] where b denotes the *context* of a. We have a designated default context $\mathbf{c} \in \mathbf{C}$. The extension to rules is done as usual: A *literal* is either an atom A *(positive literal)* or *negated atom* $(not\, A)$ *(negative literal)* and $(not\, A)^+ := A$. Let A be an atom and $L_1, \ldots, L_n$ literals $(n \geq 0)$ then $A \leftarrow L_1, \ldots, L_n$ is called *rule* with *head* A and *body* $L_1, \ldots, L_n$. A rule has to fulfil the datalog safety condition, i. e., all variables in the head atom have to occur in some positive literal of the rule body.

For more convenient usage we define (analogous to TRIPLE) some syntactical abbreviations. An *rule fragment* R is of form $(a_0 \leftarrow a_1, \ldots, a_m, L_1, \ldots, L_n)$ consisting of an f-tuple a_0, f-tuples $a_1, \ldots, a_m$ (possibly preceded by *not*) and literals $L_1, \ldots, L_n$ $(m, n \geq 0)$. The corresponding *defragmented rule wrt. to an f-tuple* b is $R^{(b)} = (a_0@b \leftarrow a_1@b, \ldots, a_m@b, L_1, \ldots, L_n)$. One can state a group of rule fragments $@b\{R_1 \ \ldots \ R_o\}$ with f-tuple $b \neq \mathbf{c}$ and $o \geq 1$. Such a group stands for the conjunction of $R_1^{(b)}, \ldots, R_o^{(b)}$ (b is also called *internal context*).

External Contexts. Access to external data sources will be provided by so-called external contexts. We distinguish between two disjoint kinds of external contexts $\bar{\mathbf{C}}, \bar{\mathbf{C}}^\vee \subseteq \mathbf{C}$. Whereas $\bar{\mathbf{C}}$, called set of *normal external contexts*, is used for integrating external relations as usual, the set of *disjunctive external contexts* $\bar{\mathbf{C}}^\vee$ is used for integrating external sources which allow for specifying disjunctions. The latter can be used for embedding OWL.

[1] As in HEX we also allow for the classical notation $t_0(t_1, \ldots, t_n)$. Furthermore unary f-tuples (x) can be abbreviated with x.

```
(SSN Prop<age, sg, child> V),(SSN favoriteCar Car)
        ← (SSN Prop V) @ families, sportyType(SSN) @ carsAndSports,
          sportyCar(Car) @ carsAndSports.

                                        @ carsAndSports {
                                            sportyType(X) ←
@ families {                                    (X hobby sport) @ rdf.
    (SSN age Age),(SSN sg Gen),             sportyCar(X) ←
    (SSN child Child) ←                         individual(X) @ owln,
        person(SSN, Age) @ db,                  sportVariant(X) @ owl
        parent(Child, SSN) @ db,            sportyCar(X) ←
        sg(SSN, Gen).                           individual(X) @ owln,
                                                roverVariant(X) @ owl.
    sg(X, Y) ←                          }
        sg(YP, XP),
        parent(X, XP) @ db,
        parent(Y, YP) @ db.
    sg(X, X) ← person(X, _) @ db.
}
```

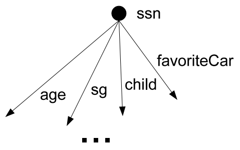


Fig. 1. Language Examples with External Contexts

The restriction for the usage is given by the following constraints: besides the datalog safety it is required that variables of atoms within disjunctive external context have to occur in body atoms set in normal external or internal contexts. We call this property *d-safety*. If the external context is given by an OWL ontology then d-safety coincides with DL-safety [5].

Fig. 1 shows an example with four external contexts. It make use of the Lloyd-Topor abbreviation stating conjunctions in the head [10]. We follow the convention that upper case letters denote variables. The program use the normal external contexts db, rdf, owln (person database, RDF base for hobbies, and car variants, respectively). The disjunctive external context owl refers to the same car variants.

The program defines two internal contexts and a rule in the default context. Context families defines triples according to person and parent from the external db context. The carsAndSports context specifies the sportiness of persons based on the RDF specification and the sportiness of cars based on the OWL ontology. Note that the owl context has to be declared as disjunctive because otherwise we would not be able to incorporate possible OWL entailments like (sportVariant $\sqcup$ roverVariant)(car524). Predicates like individual which do not require consideration of disjunctivity can be accessed via normal external contexts. Finally, the rule in default context defines triples for sporty persons based on the contexts described above and sketched by the tree fragment in the lower right corner of fig. 1.

3 Fixpoint Semantics

Before we come to the fixpoint procedure we have to consider filtered variables and context parts of atoms at the level of unification. We adopt the classical definitions of variable substitution and term unification (see also [10]) but from now on the filter part of variables is taken into account. Let $v\texttt{<}c_1, \ldots, c_n\texttt{>}$ be a filtered variable. Then a *variable substitution* θ has to fulfil $v\theta \in \{c_1, \ldots, c_n\}$ and an application of θ removes the filter part, i. e., $(v\texttt{<}c_1, \ldots, c_n\texttt{>}))\theta = c$ for $v/c \in \theta$, $c \in \mathbf{C}$. Let $(t_0 \ \ldots \ t_n)$ be an f-tuple and $a \,@\, b$ be an atom then $(t_0 \ \ldots \ t_n)\theta = (t_0\theta \ \ldots \ t_n\theta)$ and $(a \,@\, b)\theta = a\theta \,@\, b\theta$, respectively. Clearly, $a \,@\, b = c \,@\, d$ iff $a = c$ and $b = d$. The extension to *unification* and the definition of *ground* f-tuples, atoms, and programs is done as usual. Note that we stay at 'datalog level' in comparison to TRIPLE, i. e., functions symbols are not introduced.

Because negation is allowed in we have to impose restrictions to the structure of programs. On one hand, it has to be avoided that negation is used in conjunction with disjunctive contexts. On the other hand, classical stratification has to be adapted to higher-orderness. For this purposes the *dependency graph* $G_{\mathbf{P}}$ of a program $\mathbf{P}$ is defined as follows.[2] Every rule of $\mathbf{P}$ is a node. There is an arc from rule r_1 to rule r_2 if the head of r_2 is f-unifiable with the atom of some body literal L of r_1. The arc is called *negative* if L is negative, otherwise *positive*. If r_2 contains a body literal in disjunctive external context then the arc is called *disjunctive*. A program $\mathbf{P}$ is *stratified* if $G_{\mathbf{P}}$ does not contain a cyclic path with negative arcs. $\mathbf{P}$ is said to be *admissible* if is stratified and $G_{\mathbf{P}}$ does not contain a path with both a negative arc and a disjunctive arc.

The semantics of a normal external context b is given by the set of ground f-tuples provided by the external system (denoted with $Ext(b)$). They are merged into $\mathcal{E} = \{a \,@\, b \mid \ b \in \bar{\mathbf{C}}, a \in Ext(b)\}$. We furthermore assume that $\mathcal{E}$ is finite. Preparing the definition of the fixpoint procedure a construct for dealing with disjunctive external contexts has to be introduced: the expression $A \twoheadleftarrow T$ with ground atom A and a set of ground f-tuples T is called *dependent atom*. We now come to the definition when a dependent atom can be derived. Let $\mathbf{P}$ be an admissible program and $r = (A \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_n, D_1, \ldots, D_o)$ a ground rule of $\mathbf{P}$ with B_i, C_j, D_k in internal, normal external, and disjunctive external context, respectively $(m, n, o \geq 0)$. Let I be a set of dependent atoms. We say that the dependent atom

$$A \twoheadleftarrow \ \bigcup_{i=1,\ldots,m} S_i \bigcup_{j=1,\ldots,o} \{t_j\}$$

is *derivable with respect to* r *and* I if the following conditions hold

(i) B_i is positive: $(B_i \twoheadleftarrow S_i) \in I$, otherwise $(B_i^+ \twoheadleftarrow \emptyset) \notin I$ and $S_i = \emptyset$ $(i = 1, \ldots, m)$.

(ii) $C_j \in \mathcal{E}$, if C_j positive, otherwise $C_j^+ \notin \mathcal{E}$, for $j = 1, \ldots, n$.

(iii) $t_k \,@\, \mathbf{d} = D_k$ with disjunctive external context $\mathbf{d}$, for $k = 1, \ldots, o$.

[2] In the context of this paper it is sufficient to treat rules as nodes instead of considering the fine grain graph as in [3].

We allow only one disjunctive external context in (iii) for the sake of technical simplicity. This restriction can be weakened later by aggregate disjunctive contexts and treat them separately. Following the classical approaches the *immediate consequence operator* $T_{\mathbf{P}}(I)$ is defined by

$$T_{\mathbf{P}}(I) = \{X \mid r \in \mathbf{P}, \text{ and dependent atom } X \text{ is derivable wrt. } r \text{ and } I\} \cup I \ .$$

Given a partition $\mathbf{P} = \mathbf{P}^1 \cup \ldots \cup \mathbf{P}^n$ with strata $\mathbf{P}^i$ $(i = 1, \ldots, n)$ we finally compute the least fixpoint of the immediate consequence operator for every stratum in ascending order, assuming that the lowest stratum is $\mathbf{P}^1$. More precisely, let the sequence $I_1 = T^{\omega}_{\mathbf{P}^1}(\emptyset), I_2 = T^{\omega}_{\mathbf{P}^2}(I_1), \ldots, I_n = T^{\omega}_{\mathbf{P}^n}(I_{n-1})$ be given, where T^{ω}_X denotes the least fixpoint of T_X (cf. [11]),[3] the *semantics of a program* $\mathbf{P}$, denoted by $\mathit{Sem}(\mathbf{P})$, is equal I_n.

So far the procedure yields a set of dependent atoms where only normal contexts are resolved according to definition of $\mathcal{E}$. The question if a set of ground f-tuples is derivable from its disjunctive context has to be answered separately. Let $\mathbf{P}$ be a program and let $\mathcal{D}$ denote the external component which answers if a logical expression (in disjunctive normal form) over ground f-tuples is derivable, i. e., $\mathcal{D}$ has to accept $\mathit{Disj}(X_1, \ldots, X_n)$ where X_i is interpreted as a conjunction of f-tuples. Then an atom A *is derivable from* $\mathbf{P}$ if

- $(A \twoheadleftarrow \emptyset) \in \mathit{Sem}(\mathbf{P})$ or
- there exists an $k \geq 1$ such that $(A \twoheadleftarrow T_1), \ldots, (A \twoheadleftarrow T_k) \in \mathit{Sem}(\mathbf{P})$ and $\mathit{Disj}(T_1, \ldots, T_k)$ is derivable from $\mathcal{D}$.

Note that d-safety, which is DL-safety for arbitrary disjunctive components, guarantees grounding. In the case of description logic special transformations like folding, as described in [7], could be performed during the derivation step defined by (i)-(iii) above.

4 Pragmatics

One of the properties an ontology language (with rules) should have is the ability to express transformations. Generally, every representation of information in form of sets of logical facts and rules can be the source for such transformations. An example from the automotive area is described in [12]. Transformations in the area of UML/MDA are described in [13][14]. As already shown in Figure 1 the rule in default context can be viewed as an transformation rule for the facts which are valid in internal contexts families and carsAndSports. Of course, one can build compositions of transformations by chaining contexts where the top context can be parameterized. But what we really want is to be able to parameterize the whole chain. Let us assume the following abstract example of two contexts.

[3] It exists due to the fact that the set of dependent atoms together with $\subseteq$ forms a complete lattice and T_X is monotone wrt. a stratum.

$$@\ f(X)\{r(\ldots) \leftarrow r(\ldots) @ X,\ \bar{p}\ .\}$$

$$@\ g(X)\{r(\ldots) \leftarrow r(\ldots) @ X.\ \bar{q}\ .\}$$

The first context defines the r-relation[4] based on the context stated as a parameter X and modified according to remaining atoms $\bar{p}$. The second context does the same but with $\bar{q}$ instead. If a transformation combining both contexts is needed the following expression can be stated.

$$\leftarrow r(\ldots) @ s_1 \ \ \textsf{where} \ \ s_1 := r(\ldots) @ f(s_2),\ s_2 := r(\ldots) @ g(a).$$

It first builds the context s_2 by querying $r(\ldots) @ g(a)$ and then s_1 is constructed by querying $r(\ldots) @ f(s_2)$. This mechanism enables the composition of arbitrary contexts for the purpose of fact transformations without skolemization based on function symbols.

A first partial implementation of the language is realized under the name JORL[5]. Our support for filtered variables as a language construct suggested the use of static filtering [15]. The filter propagation algorithm was slightly adapted to the higher-orderness of rules and to filtered variables.

5 Related Work

Our approach was mainly influence by TRIPLE [8]. One of the strengths of TRIPLE is the ability to express transformations by parametrization of contexts. TRIPLE realizes it by using terms as context expressions and allowing skolemization based on function symbols. JORL aimed at staying at datalog level where context parts are expressed by f-tuples which are essentially classical function-free, (syntactical) higher-order atoms. To compose parameterized contexts we use where-chains instead of (chained) skolemization. The context approach is also comparable with the F-Logic implementation FLORA2 [16] and its module concept, but there only constants are allowed to define modules although it has to be noted that discussions are ongoing[6] to allow first-order terms but no higher-orderness. JORL does not support reification as TRIPLE and FLORA2 but offer in addition disjunctive external contexts.

Concerning external components JORL is influenced by the language HEX [3]. HEX offers higher expressivity due to the fact that disjunctive datalog with non-monotonic negation builds the base. Moreover, external components (or external predicates) can retrieve program predicate extensions which make HEX a homogenous language. In contrast to JORL it does not supportcontexts.

[4] An concrete example could be the *triple*-relation in conjunction with RDF.

[5] Jönköping Rule Language
(demo at http://hem.hj.se/~bill/Jot/Jorl/Tryout/JORLTryoutApplet.html);
planned as a part of the Jönköping Ontology Toolkit JOT
(http://hem.hj.se/~bill/Jot/Site/jot-test-1.html).

[6] http://forum.projects.semwebcentral.org/forum-modules.html.

Furthermore, full external disjunctivity is unsupported, i. e., given the rules $a \leftarrow p$ and $a \leftarrow q$ (p, q external predicates) atom a cannot be derived if disjunction $p \vee q$ is valid.

KAON2 [5] enables DL-reasoning by transformation to disjunctive datalog. As our approach it requires DL-safety. In contrast to our approach KAON2 fully integrates rules and DL just like DL+Log [6] which extends KAON2 with non-monotonic negation and weak safety (existential variables in DL-atoms). However, it has no support for contexts and higher-orderness, but it has to be remarked that higher-orderness could be introduced by [17].

Finally, Datalog^{DL} [7] is an hybrid language extending the AL+log [18] approach. Concerning the reasoning procedure it is very similar to Jorl but it bases on classical SLD-resolution instead of bottom-up evaluation which do not allow for recursive programs. Contexts, higher-orderness, and negation are not supported. Its strength is given by the fact that no extra safety condition apart from datalog safety is required. An elaborate handling of DL expressions during the reasoning process is proposed which allows for having existential variables in DL atoms. External components are supported in order to solve DL queries.

6 Future Work

Possible directions for future work are manifold. Concerning disjunctivity the approach should be extended by dealing with optimizations and transformations on the level of disjunctive expressions itself (in the sense of [7]). There is also the question which other kinds of disjunctive systems apart from DL or propositional logic in the direction of [19] could be incorporated. Concerning contexts it is of interest wether Jorl could be extended to serve as an model transformation language in the area of UML whose models be easily represented by fact sets in the sense of [13]. Also our restrictions in using non-monotonic negation could be weakened in order to use it in conjunction with disjunctive external contexts. Finally, we want to investigate to which extent the hybridness of Jorl can be weakened, leading to so-called bidirectional contexts.

Acknowledgements

This work is aligned to the research project DEON (Development and Evolution of Ontologies in Networked Organizations) based on a grant from STINT (The Swedish Foundation for International Cooperation in Research and Higher Education; grant IG 2008-2011).

References

1. Chen, W., Kifer, M., Warren, D.S.: HILOG: A foundation for higher-order logic programming. Journal of Logic Programming 15(3), 187–230 (1993)
2. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of ACM 42(4), 741–843 (1995)

3. 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)
4. Staab, S., Studer, R. (eds.): Handbook on Ontologies. International Handbooks on Information Systems. Springer, Heidelberg (2004)
5. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Universität Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe (2006)
6. Rosati, R.: Dl+log: Tight integration of description logics and disjunctive datalog. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) KR 2006, pp. 68–78. AAAI Press, Menlo Park (2006)
7. Mei, Jing, Lin, Zuoquan, Boley, Harold, Li, Jie, Bhavsar, Virendrakumar, C.: The datalog dl combination of deduction rules and description logics. The datalog dl combination of deduction rules and description logics 23(3), 356–372 (2007)
8. Sintek, M., Decker, S.: TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342. Springer, Heidelberg (2002)
9. McCarthy, J.: Notes on formalizing contexts. In: Kehler, T., Rosenschein, S. (eds.) Proceedings of the Fifth National Conference on Artificial Intelligence, Los Altos, California, pp. 555–560. Morgan Kaufmann, San Francisco (1993)
10. Lloyd, J.W.: Foundations of Logic Programming. Springer, New York (1993)
11. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
12. Sandkuhl, K., Billig, A.: Ontology-based artefact management in automotive electronics. International Journal of Computer Integrated Manufacturing 20(7), 627–638 (2007)
13. Billig, A., Busse, S., Leicher, A., Süß, J.G.: Platform Independent Model Transformation Based on Triple. In: Middleware 2004, ACM/IFIP/USENIX International Middleware Conference, pp. 493–512 (2004)
14. Decker, S., Sintek, M., Billig, A., Henze, N., Dolog, P., Nejdl, W., Harth, A., Leicher, A., Busse, S., Ambite, J.L., Weathers, M., Neumann, G., Zdun, U.: Triple - an rdf rule language with context and use cases. In: Rule Languages for Interoperability, W3C (2005)
15. Kifer, M., Lozinskii, E.L.: On compile-time query optimization in deductive databases by means of static filtering. ACM Trans. Database Syst. 15(3), 385–426 (1990)
16. Yang, G., Kifer, M., Zhao, C.: Flora-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 671–688. Springer, Heidelberg (2003)
17. Motik, B.: On the Properties of Metamodeling in OWL. In: International Semantic Web Conference, pp. 548–562 (2005)
18. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: AL-log: Integrating Datalog and Description Logics. J. Intell. Inf. Syst. 10(3), 227–252 (1998)
19. Goasdoue, F., Rousset, M.C.: Querying distributed data through distributed ontologies: A simple but scalable approach. IEEE Intelligent Systems 18(5), 60–65 (2003)

Paraconsistent Reasoning with Quasi-classical Semantic in $\mathcal{ALC}$[⋆]

Xiaowang Zhang[1,2] and Zuoquan Lin[1]

[1] Department of Information Science, Peking University, China
[2] School of Mathematical Sciences, Anhui University, China
{zxw,lzq}@is.pku.edu.cn

Abstract. Description logics are a family of knowledge representation formalism which descended from semantic networks. During the past decade, the important reasoning problems such as satisfiability and subsumption have been handled by tableau-like algorithms. Description logics are practical monotonic logics which, though imparting strong and conclusive reasoning mechanisms, lack the flexibility of non-monotonic reasoning mechanisms. In recent years, the study of inconsistency handling in description logics becomes more and more important. Some technologies are being applied to handle inconsistency in description logic. Quasi-classical logic, which allows the derivation of nontrivial classical inferences from inconsistent information, supports many important proof rules such as modus tollens, modus ponens, and disjunctive syllogism. In this paper, we consider the characters of $\mathcal{ALC}$ with Quasi-classical semantics and develop a sound and complete tableau algorithm for paraconsistent reasoning in $\mathcal{ALC}$.

1 Introduction

Paraconsistent reasoning is important in handling inconsistent information, and there have been a number of proposals for paraconsistent logics(for a review see [5]). There are also many different approaches and technologies for handling inconsistency in description logics. For example, key paraconsistent logics such as C_ω [4] achieve non-trivializable property by weakening the classical connectives, particularly negation. $\mathcal{ALC}4$ [11] yields to the insight that inconsistency is natural phenomenon in realistic data and supports the non-trivializable property. However they fail in useful proof rules such as disjunctive syllogism and intuitive equivalences, i.e., $\neg\alpha \vee \beta \equiv \alpha \rightarrow \beta$.

Quasi-classical (or QC) logic restricts the proof theory [3,7]. In this restriction, decompositional rules cannot follow compositional proof rules. Reasoning with inconsistencies arising in applications is a very important direction. For example, there are some technologies such as systems development [6] and reasoning with structured text [8]. The first-order logic version of paraconsistent logic and the semantic tableau for first-order QC logic have been discussed by Anthony Hunter [9]). And Anthony Hunter provides a general characterization of inconsistency, based on quasi-classical

⋆ The authors gratefully acknowledge the support of Major Program of National Natural Science Foundation of China(60496322).

logic (a form of paraconsistent logic with a more expressive semantics than Belnaps four-valued logic [2], and unlike other paraconsistent logics, allows the connectives to appear to behave as classical connectives) [10].

In this paper, we present a description logic version of paraconsistent logic. First we give strong and weak semantic for $\mathcal{ALC}$ and then develop a semantic tableau of the proof theory, called QC semantic tableau which is used to handle instance checking.

The contents of this paper are summarized as follows. In Section 2, $\mathcal{ALC}$ as a basic knowledge representation is introduced. In Section 3, the definition of QC entailment is introduced which is help handle an ontology with inconsistency in $\mathcal{ALC}$. In Section 4, we introduce a QC semantic tableau in order to handle instance checking of an ontology with only $ABox$. And we conclude terminability, soundness and completeness of QC semantic tableau in instance checking. In Section 5, we conclude our works. Due to space limitation, some proofs are only sketched, and their detailed proofs can be found at `http://www.is.pku.edu.cn/~zxw/QCTableau.pdf`.

2 The Description Logic $\mathcal{ALC}$

We briefly review notation and terminology of the description logic $\mathcal{ALC}$, but we basically assume that the reader is familiar with description logics. For comprehensive background reading, please refer to [1].

Given a set of atomic concepts (or concept names), a set of roles (or role names), and a set of individuals,special symbols $\top$ and $\bot$ refer to the top concept and the bottom concept, respectively. Complex concepts in $\mathcal{ALC}$ can be formed from these inductively as follows.

1. $\top$, $\bot$, and each atomic concept are concepts;
2. If C, D are concepts, then $C \sqcup D$, $C \sqcap D$, and $\neg C$ are concepts;
3. If C is a concept and R is a role, then $\forall R.C$ and $\exists R.C$ are concepts.

An $\mathcal{ALC}$ ontology consists of a set of assertions, called the $ABox$ of the ontology, and a set of inclusion axioms, called the $TBox$ of the ontology. Assertions are of the form $C(a)$ or $R(a, b)$, where a, b are individuals and C and R are concepts and roles, respectively. Inclusion axioms are of the form $C \sqsubseteq D$, where C and D are concepts. Informally, an assertion $C(a)$ means that the individual a is an instance of concept C, and an assertion $R(a, b)$ means that individual a is related with individual b via the property R. The inclusion axiom $C \sqsubseteq D$ means that each individual of C is an individual of D.

The formal definition of the (model-theoretic) semantics of $\mathcal{ALC}$ is given by means of interpretations $I = (\Delta^I, \cdot^I)$ consisting of a non-empty domain Δ^I and a mapping $\cdot^I$ satisfying the conditions in Table 1, interpreting concepts as subsets of the domain and roles as binary relations on the domain. An interpretation satisfies an $\mathcal{ALC}$ ontology (i.e. is a model of the ontology) iff it satisfies each axiom in both the $ABox$ and the $TBox$. An ontology is called satisfiable (unsatisfiable) iff there exists (does not exist) such a model. In $\mathcal{ALC}$, reasoning tasks, i.e. the derivation of logical consequences, can be reduced to satisfiability checking of ontologies in [1].

Table 1. Syntax and semantics of $\mathcal{ALC}$

Syntax	**Semantics**
A	$A^I \subseteq \Delta^I$
R	$R^I \subseteq \Delta^I \times \Delta^I$
o	$o^I \in \Delta^I$
$\top$	Δ^I
$\bot$	$\emptyset$
$C_1 \sqcap C_2$	$C_1^I \cap C_2^I$
$C_1 \sqcup C_2$	$C_1^I \cup C_2^I$
$\neg C$	$\Delta^I \setminus C^I$
$\exists R.C$	$\{x \mid \exists y, (x,y) \in R^I$ and $y \in C^I\}$
$\forall R.C$	$\{x \mid \forall y, (x,y) \in R^I$ implies $y \in C^I\}$

Syntax	**Semantics**
$C_1 \sqsubseteq C_2$	$C_1^I \subseteq C_2^I$
$C(a)$	$a^I \in C^I$
$R(a,b)$	$(a^I, b^I) \in R^I$

3 QC Description Logic

QC description logic is a development of QC logic [7].

Definition 1. *Let A be a primitive concept, and let $\sim$ be a complementation operation such that $\sim A$ is $\neg A$ and $\sim (\neg A)$ is A.*

The $\sim$ operator is not a part of the object language, but it makes some definitions clearer.

Definition 2 (Focus). *Let $A_1 \sqcup \cdots \sqcup A_n$ be a concept that includes a literal disjunct A_i. The focus of $A_1 \sqcup \cdots \sqcup A_n$ by A_i, denoted $\otimes(A_1 \sqcup \cdots \sqcup A_n, A_i)$ is defined as the concept obtained by removing A_i from $A_1 \sqcup \cdots \sqcup A_n$. In the case of a clause with just one disjunct, we assume $\otimes(A_1, A_1) = \bot$.*

In the following, let S be a subset of Δ^I and we write $\overline{S} = \Delta^I \backslash S$.

The notion of a model in QC description logic is based on the QC interpretation.

Definition 3 (Strong Interpretation). *A strong interpretation I is a pair $I = (\Delta^I, \cdot^I)$ with Δ^I as domain, where $\cdot^I$ is a function assigning elements of Δ^I to individuals, and subsets of $(\Delta^I)^2$ to concepts, such that the conditions in Table 2 are satisfied.*

Definition 4 (Strong Satisfaction). *For a strong interpretation I in $\mathcal{ALC}$, we define strong satisfaction $\models_s$ as follows, where $A_1, \ldots, A_n, A$ are literals, R is a role and an individual a.*

$I \models_s A(a)$ iff $a \in +A$, where $A^I = \langle +A, -A \rangle$
$I \models_s R(a,b)$ iff $(a,b) \in +R$, where $R^I = \langle +R, -R \rangle$
$I \models_s (C \sqcap D)(a)$ iff $I \models_s C(a)$ and $I \models_s D(a)$

Table 2. QC Strong Semantics of Concepts

Syntax	**Strong Semantics**
A	$A^I = \langle +A, -A \rangle$,
	where $+A, -A \subseteq \Delta^I$
R	$R^I = \langle +R, -R \rangle$,
	where $+R, -R \subseteq \Delta^I \times \Delta^I$
o	$o^I \in \Delta^I$
$\top$	$\langle \Delta^I, \emptyset \rangle$
$\bot$	$\langle \emptyset, \Delta^I \rangle$
$C_1 \sqcap C_2$	$\langle +C_1 \cap +C_2, (-C_1 \cup -C_2)$
	$\cap(-C_1 \cup \overline{+C_2}) \cap (\overline{+C_1} \cup -C_2) \rangle$,
	if $C_i^I = \langle +C_i, -C_i \rangle$ for $i = 1, 2$,
$C_1 \sqcup C_2$	$\langle (+C_1 \cup +C_2) \cap (\overline{-C_1} \cup +C_2)$
	$\cap(+C_1 \cup \overline{-C_2}), -C_1 \cap -C_2 \rangle$,
	if $C_i^I = \langle +C_i, -C_i \rangle$ for $i = 1, 2$,
$\neg C$	$(\neg C)^I = \langle -C, +C \rangle$, if $C^I = \langle +C, -C \rangle$
$\exists R.C$	$\langle \{x \mid \exists y, (x, y) \in +R$ and $y \in +C\}$,
	$\{x \mid \forall y, (x, y) \in +R$ implies $y \in -C\} \rangle$
$\forall R.C$	$\langle \{x \mid \forall y, (x, y) \in +R$ implies $y \in +C\}$,
	$\{x \mid \exists y, (x, y) \in +R$ and $y \in -C\} \rangle$

Syntax	**Semantics**
$C \sqsubseteq D$	$+C \subseteq +D$,$-D \subseteq -C$
	where $C^I = \langle +C, -C \rangle$,$D^I = \langle +D, -D \rangle$
$C(a)$	$a \in +C, C^I = \langle +C, -C \rangle$
$R(a, b)$	$(a, b) \in +R, R^I = \langle +R, -R \rangle$

$I \models_s C \sqsubseteq D$ iff $I \models_s C(a)$ implies $I \models_s D(a)$ and $I \models_s \neg D(a)$ implies $I \models_s \neg C(a)$

$I \models_s (A_1 \sqcup \cdots \sqcup A_n)(a)$ iff $[I \models_s A_1(a)$ or $\ldots$ or $I \models_s A_n(a)]$ and $\forall i \,.\, 1 \leq i \leq n$ $[I \models_s \sim A_i(a)$ implies $I \models_s \otimes((A_1 \sqcup \cdots \sqcup A_n)(a), A_i(a))]$.

Definition 5 (Weak Interpretation). *Weak interpretation I is a pair $I = (\Delta^I, \cdot^I)$ with Δ^I as domain, where $\cdot^I$ is a function assigning elements of Δ^I to individuals, and subsets of $(\Delta^I)^2$ to concepts, such that the conditions in Table 3 are satisfied.*

The definition for weak satisfaction is similar to strong satisfaction. The main difference is that the definition of disjunction is less restricted.

Definition 6 (Weak Satisfaction). *For a weak interpretation I , we define weak satisfaction $\models_w$ as follows, where $A_1, \ldots, A_n, A$ are literals, R is a role and an individual a.*

$I \models_w A(a)$ iff $a \in +A$
$I \models_w R(a, b)$ iff $(a, b) \in +R$
$I \models_w (C \sqcap D)(a)$ iff $I \models_w C(a)$ and $I \models_w D(a)$
$I \models_w C \sqsubseteq D$ iff $I \models_w C(a)$ implies $I \models_w D(a)$

Table 3. QC Weak Semantics of Concepts

Syntax	Weak Semantics
A	$A^I = \langle +A, -A \rangle$, where $+A, -A \subseteq \Delta^I$
R	$R^I = \langle +R, -R \rangle$, where $+R, -R \subseteq \Delta^I \times \Delta^I$
o	$o^I \in \Delta^I$
$\top$	$\langle \Delta^I, \emptyset \rangle$
$\bot$	$\langle \emptyset, \Delta^I \rangle$
$C_1 \sqcap C_2$	$\langle +C_1 \cap +C_2, -C_1 \cup -C_2 \rangle$, if $C_i^I = \langle +C_i, -C_i \rangle$ for $i = 1, 2$
$C_1 \sqcup C_2$	$\langle +C_1 \cup +C_2, -C_1 \cap -C_2 \rangle$, if $C_i^I = \langle +C_i, -C_i \rangle$ for $i = 1, 2$
$\neg C$	$(\neg C)^I = \langle -C, +C \rangle$, if $C^I = \langle +C, -C \rangle$
$\exists R.C$	$\langle \{x \mid \exists y, (x, y) \in +R$ and $y \in +C\}$, $\{x \mid \forall y, (x, y) \in +R$ implies $y \in -C\} \rangle$
$\forall R.C$	$\langle \{x \mid \forall y, (x, y) \in +R$ implies $y \in +C\}$, $\{x \mid \exists y, (x, y) \in +R$ and $y \in -C\} \rangle$

Syntax	Semantics
$C \sqsubseteq D$	$+C \subseteq +D$,where $C^I = \langle +C, -C \rangle$, $D^I = \langle +D, -D \rangle$
$C(a)$	$a \in +C, C^I = \langle +C, -C \rangle$
$R(a, b)$	$(a, b) \in +R, R^I = \langle +R, -R \rangle$

$I \models_w (A_1 \sqcup \cdots \sqcup A_n)(a)$ *iff* $[I \models_w A_1(a)$ *or* $\ldots$ *or* $I \models_w A_n(a)]$.

In the following, we introduce two QC models: strong QC model and weak QC model in order to define QC entailment.

Definition 7 (Strong QC Model). *Let $I = (\Delta^I, \cdot^I)$ and O be an ontology, i.e., $O = (ABox, TBox)$. I is a strong QC model of O, denoted by $I \models_s O$ iff for any $C \sqsubseteq D \in TBox$ and for any $C(a), R(a, b) \in ABox$, $I \models_s C \sqsubseteq D$, $I \models_s C(a)$ and $I \models_s R(a, b)$.*

Definition 8 (Weak QC Model). *Let $I = (\Delta^I, \cdot^I)$ and O is an ontology, i.e., $O = (ABox, TBox)$. I is a weak QC model of O, denoted by $I \models_w O$ iff for any $C \sqsubseteq D \in TBox$ and for any $C(a), R(a, b) \in ABox$, $I \models_w C \sqsubseteq D$, $I \models_w C(a)$ and $I \models_w R(a, b)$.*

Definition 9 (QC Entailment). *Let $\models_{QC}$ be an entailment relation, called QC entailment relation, defined as follows:*

$$\{\alpha_1, \ldots, \alpha_n\} \models_{QC} \beta \text{ iff for every interpretation } I$$
$$\text{if } I \models_s \alpha_i \text{ for all } i \ (1 \leq i \leq n) \text{ then } I \models_w \beta.$$

where α_i, β are formulae in $ABox$.

We can show that $\models_{QC}$ is non-trivializable in the sense that when O is classically inconsistent, it is not the case every formula in $\mathcal{ALC}$ is entailed by $\models_{QC}$.

Example 1. *Let $O = \{B(a), \neg B(a)\}$ and $A(a)$ be primitive concepts in $\mathcal{ALC}$. So $B(a) \sqcap \neg B(a)$ is classically inconsistent. However it is not the case that $O \models_{QC} A(a)$ holds, since $O \models_{QC} (B \sqcap \neg B)(a)$, but $O \models_{QC} A(a)$.*

Example 2. *Let $O = \emptyset$. Now consider the classical $\top = (A \sqcup \neg A)(a)$. Here $O \models_{QC} (A \sqcup \neg A)(a)$ does not hold. Since $\emptyset$ strongly satisfies every formula in O, but $\emptyset$ does not weakly satisfy $(A \sqcup \neg A)(a)$.*

4 Tableau Algorithm for Quasi-classical $\mathcal{ALC}$

Definition 10 (Instance Checking). *Let $\mathcal{O}$ is an ontology, C is a concept and a is an individual in $\mathcal{ALC}$. Instance checking is a question to check whether a is an instance of C or not under the ontology $\mathcal{O}$, i.e., $\mathcal{O} \models_{QC} C(a)$.*

Definition 11 (Subsumption). *Let $\mathcal{O}$ is an ontology and C, D is a concept, for all interpretation I, if $I \models_s \mathcal{O}$ and $C^I \subseteq D^I$, i.e., $I \models_s O$ and $I \models_w C \sqsubseteq D$, then C is subsumed by D under the ontology $\mathcal{O}$, denoted by $O \models_{QC} C \sqsubseteq D$.*

In order to provide an automated proof procedure, we adapt the tableau approach for $\mathcal{ALC}$ that was developed by introducing some new transformation rules.

Definition 12. *The set of signed formulae of $\mathcal{ALC}$ is denoted $\mathcal{ALC}^*$ and is defined as $\mathcal{ALC} \cup \{\alpha^* | \alpha \in \mathcal{ALC}\}$.*

We will regard the formulae in $\mathcal{ALC}^*$ without the $*$ symbol as satisfiable and the formulae in $\mathcal{ALC}^*$ with the $*$ symbol as unsatisfiable.

Definition 13. *We further extend the weak satisfaction and strong satisfaction relations as follows where C, D is a concept and a is an individual in $\mathcal{ALC}$.*

$$I \models_s C^*(a) \textit{ iff } I \not\models_s C(a), I \models_w C^*(a) \textit{ iff } I \not\models_w C(a)$$

$$I \models_s (C \sqsubseteq D)^* \textit{ iff } I \not\models_s C \sqsubseteq D, I \models_w (C \sqsubseteq D)^* \textit{ iff } I \not\models_w C \sqsubseteq D$$

It will be convenient to assume that all concept descriptions are in negation normal form (NNF), i.e., that negation occurs only directly in front of concept names.

Definition 14 (Quasi-Classical Transformation Rule). *Let an ontology $\mathcal{O}$ be $(\mathcal{A}, \mathcal{T})$; C_1, C_2, C be concepts; R be role and x, y, z be individuals. Quasi-classical (QC) transformation rules are defined in Table 4.*

Proposition 1 (Soundness). *Let $\mathcal{A}$ be an $ABox$ and $\mathcal{A}'$ be obtained from $\mathcal{A}$ by using a QC transformation rule in each step. Then $\mathcal{A}$ is consistent iff $\mathcal{A}'$ is consistent.*

Definition 15 (Complete). *An $ABox$ $\mathcal{A}$ is called complete iff none of the QC transformation rules of Table 4 applies to it.*

Table 4. QC Transformation Rules in $\mathcal{ALC}$

The $\rightarrow_{\sqcap}$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(x), C_2(x)\}$.
The $\rightarrow_{\exists}$-rule
Condition: $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual name z such that $C(z)$ and $R(x, z)$ are in $\mathcal{A}$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(y), R(x, y)\}$ where y is an individual name not occurring in $\mathcal{A}$.
The $\rightarrow_{\forall}$-rule
Condition: $\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x, y)$, but not contain $C(y)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$.
The $\rightarrow_{QC}$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$ and $\sim C_1(x)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_2(x)\}$.
The $\rightarrow_{\sqcap}^{*}$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)^*(x)$, but not both $C_1^*(x)$ and $C_2^*(x)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1^*(x)\}, \mathcal{A}'' := \mathcal{A} \cup \{C_2^*(x)\}$.
The $\rightarrow_{\sqcup}^{*}$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)^*(x)$, but not both $C_1^*(x)$ and $C_2^*(x)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1^*(x), C_2^*(x)\}$.
The $\rightarrow_{\exists}^{*}$-rule
Condition: $\mathcal{A}$ contains $(\exists R.C)^*(x)$ and $R(x, y)$, but not contain $C^*(y)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C^*(y)\}$.
The $\rightarrow_{\forall}^{*}$-rule
Condition: $\mathcal{A}$ contains $(\forall R.C)^*(x)$, but there is no individual name z such that $C^*(z)$ and $R(x, z)$ are in $\mathcal{A}$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C^*(y), R(x, y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

Definition 16 (Closed). *An $ABox$ $\mathcal{A}$ contains a clash iff $\{C(x), C^*(x)\} \subseteq \mathcal{A}$ for some concept C and for some individual x. An $ABox$ $\mathcal{A}$ is called closed if it contains a clash, and open otherwise.*

Proposition 2 (Completeness). *If an $ABox$ $\mathcal{A}$ is complete and closed then $\mathcal{A}$ is inconsistent.*

Proposition 3 (Terminability). *The tableau algorithm terminates after finite steps using QC transformation rules, i.e., there cannot be an infinite sequence of rule application.*

So a finite sequence denoted by $\mathcal{S}$ of $ABox$s, which is $(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$, is obtained from $\mathcal{A}$ by using QC transformation rules every step. We conclude that the tableau algorithm for QC $\mathcal{ALC}$ is sound and complete.

Theorem 1. *Let $\mathcal{A}$ be an $ABox$ and α be a query. $\mathcal{A}'$, which is obtained from $\mathcal{A} \cup \{\alpha^*\}$ by using the tableau algorithm, is closed iff $\mathcal{A}$ entails α under the QC semantic, i.e., $\mathcal{A} \models_{QC} \alpha$.*

In the following, we only consider the question about instance checking in $ABox$, that is, $\mathcal{T}$ can be regarded as the empty set.The problem of instance checking whether a is an instance of C under QC semantics or not can be transformed into the problem whether $\mathcal{A}$ entails $C(a)$ or not. So we can reduce the instance checking to the consistency problem for $ABox$ because there is the following connection.

Corollary 1. *Let $\mathcal{A}$ be an $ABox$, C be a concept and a be an individual in $\mathcal{ALC}$. $\mathcal{A} \models_{QC} C(a)$ iff $\mathcal{A} \cup \{C^*(a)\}$ is inconsistent.*

5 Conclusions

In this paper, we have introduced QC into DLs in order to handle $\mathcal{ALC}$ with inconsistency. We have defined QC DLs and its strong semantic and weak semantics. By using strong interpretation $\models_s$ and weak interpretation $\models_w$, QC entailment $\models_{QC}$ has been depicted naturally. In the second part, we have developed a new tableau called by QC semantic tableau, which is different from classical tableaus of DLs to handle instance checking an ontology $\mathcal{O}$ with only $ABox$, i.e., its $Tbox$ is null. We have concluded terminability, soundness and completeness of QC semantic tableau in instance checking.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
2. Belnap, N.D.: A useful four-valued logic. Modern uses of multiple-valued logics, 7–73 (1977)
3. 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)
4. da Costa, N.C.: On the theory of inconsistent formal systems. Notre Dame Journal of Formal Logic 15, 496–510 (1974)
5. Hunter, A.: Paraconsistent logics. Handbook of Defeasible Reasoning and Uncertainty Management Systems 2, 11–36 (1998)
6. Hunter, A., Nuseibeh, B.: Managing inconsistent sepecifiations:reasoning, analysis and action. Transactions on Software Engineering and Methpdplogy 7, 335–367 (1998)
7. Hunter, A.: Reasoning with contradictory information using quasi-classical logic. J. Log. Comput. 10(5), 677–703 (2000)
8. Hunter, A.: Reasoning with inconsistency in structured text. Knowledge Engineering Review 15, 317–337 (2000)
9. Hunter, A.: A semantic tableau version of first-order quasi-classical logic. In: Benferhat, S., Besnard, P. (eds.) ECSQARU 2001. LNCS (LNAI), vol. 2143, pp. 544–555. Springer, Heidelberg (2001)
10. Hunter, A.: Measuring inconsistency in knowledge via quasi-classical models. Springer, Heidelberg (2002)
11. Ma, P.H.Y., Lin, Z.: Paraconsistent resolution for four-valued description logics (2007)

Fluent Calculus Based Web Service Composition

Viorica Rozina Chifu, Ioan Salomie, and Simona Manole

Technical University of Cluj-Napoca, Department of Computer Science, Bariţiu 28
{Viorica.Chifu,Ioan.Salomie}@cs.utcluj.ro

1 Introduction and Related Work

This paper presents a novel approach for semantic web service composition based on the formalism of fluent calculus. We show how the planning capabilities of the *fluent calculus* can be used to automatically generate an *abstract composition model*. For describing the capabilities of web services we have used an OWL-S ontology. Based on the OWL-S ontology semantics, we encode the web service description in the *fluent calculus* formalism and provide a planning strategy for service composition. To test our composition method, we have implemented an experimental framework that automatically composes and executes web services. Our approach is similar with McIlraiths [2] in terms of computational model, both approaches having the roots in the theory of situation calculus, but the two solutions are complementary. In the situation calculus, the successor state axioms describe how a particular fluent may be changed by an action, whereas the state update axioms of the fluent calculus describe which fluents are changed by an action. The two approaches also differ in the way to evaluate the conditions in the programming languages (GOLOG and FLUX) that implement the two formalisms. For condition evaluation, GOLOG applies the principle of regression, while FLUX [3] uses the principle of progression. Using regression is efficient for short action sequences, but the computational effort increases with the number of performed actions, while by using progression, the computational effort remains the same, independently of the number of performed actions.

2 Web Service Composition with the Fluent Calculus

A fluent calculus planning problem can be described, like in the classical planning, by the (S, G, A, PE) tuple, where S is the *initial state* of the world, G is the *goal* to be achieved, A is a set of possible actions which may be executed to achieve the goal, and PE is a set of *preconditions* and *effects*. In the fluent calculus, the preconditions are encoded as action precondition axioms specifying the necessary conditions for executing action a in state S. The fluent calculus *effects* are encoded as state update axioms. Based on these assumptions, we have developed the *BkComposition* algorithm as a fluent calculus composition algorithm using backtracking search. The composition algorithm starts by considering the input parameters encoded as a list of fluents that represent the initial state Z_0, and tries to reach the goal state Z which contains all the user specified outputs.

In every intermediate state, the algorithm performs two important operations: (1) it verifies whether the preconditions of an action in this current state are satisfied in order to include the action in the compositional plan, and (2) the effect of the included action is applied to the current state in order to produce a state transition. The fluent calculus knowledge base used in the planning process is automatically generated from the OWL-S service descriptions. The mapping rules to automatically translate an OWL-S atomic process into an equivalent fluent calculus service specification are presented in Table 1. An OWL-S composite process can also be translated into the formalism of fluent calculus, a translation scheme for composite processes being presented in one of our previous work [1]. To validate our approach for automatic web service composition based on the fluent calculus formalism, we have developed an experimental prototype that integrates the translation algorithm, the fluent calculus planning technique, a user interface and an execution engine. The framework has been tested on a *trip planning* scenario and on an *eBookshop processing* scenario.

Table 1. The translation of OWL-S atomic process into the fluent calculus

OWL-S concepts	Fluent calculus concepts	Comments
Atomic Process	Action	
Inputs	Inputs and knowledge preconditions	Input parameters of the actions and action precondition axioms
Outputs	Knowledge effects	State update axioms
Preconditions	Physical preconditions	Action precondition axioms
Effects	Physical effects	State update axioms

3 Conclusions

In this paper we have presented a new approach for automatic web service composition based on the fluent calculus. The main contributions of our work consist of the translation algorithm from the OWL-S semantic web service descriptions into the fluent calculus formalism, and a fluent calculus based planning strategy.

References

1. Chifu, V., Salomie, I., Chifu, E.S.: Fluent calculus-based Web service composition - A mapping between OWL-S and fluent calculus. In: ICCP, Cluj-Napoca (2008)
2. McIlraith, S., Son, T.: Adapting Golog for Composition of Semantic Web Services. In: The Eighth International Conference on Knowledge Representation and Reasoning (KR 2002), Touluse, pp. 482–493 (2002)
3. Thielscher, M.: Programming of Reasoning and Planning Agents with FLUX. In: The International Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann, San Francisco (2002)

Vague-SWRL: A Fuzzy Extension of SWRL

Xing Wang, Z.M. Ma, Li Yan, and Xiangfu Meng

College of Information Science and Engineering, Northeastern University, Shenyang, Liaoning 110004, China
xingwang_neu@yahoo.cn, mazongmin@ise.neu.edu.cn

1 Introduction

Rules in the Web have become a mainstream topic since inference rules are marked up for e-commerce and are identified as a design issue of the semantic web [7]. SWRL [4] is incapable of representing the imprecision and uncertainty, and a single membership degree in fuzzy sets [6] is inaccurate to represent the imprecise knowledge. Based on vague sets [2] which employ membership degree intervals to represent fuzzy information, we propose a fuzzy extension of SWRL—vague-SWRL. What's more, weights in f-SWRL [5] have no power to represent the importance of membership degrees. In order to modify the membership degrees and to balance and supplement the weights of vague classes and properties (i.e., first degree weights), we present the notion of second degree weight to represent weights of the membership degrees in vague-SWRL. In addition, we extend RuleML to express vague-SWRL.

2 Vague-SWRL

Vague rules are of the form antecedent→consequent, where antecedent has the form of conjunctions of atoms, and consequent is an atomic consequent. And all the atoms can have weights (i.e., numbers between 0 and 1). The general form of vague rule is as follows:

$$\bigwedge_{i=0\ldots n}\left[vc_i{*}fdw_i\right]_{\left[vcv_i{*}sdw_i\right]} \wedge \bigwedge_{j=0\ldots n}\left[vp_j{*}fdw_j\right]_{\left[vpv_j{*}sdw_j\right]} \rightarrow vc*w\ (\,or\ vp*w\,)\ . \tag{1}$$

Where $\bigwedge_{i=0\ldots n}\left[vc_i{*}fdw_i\right]$ and $\bigwedge_{j=0\ldots n}\left[vp_j{*}fdw_j\right]$ are conjunctions of vague classes and vague properties, respectively. *vc*, *vp*, *w* and *fdw* stand for vague class, vague property, weight, and first degree weight, respectively. When given membership degrees of vague classes and properties, second degree weights are correspondingly obtained to modify these membership degrees and to balance first degree weights. *vcv*, *sdw* and *vpv* are vague class value, second degree weight and vague property value, respectively. Because the membership degree values of vague classes and properties are represented by membership degree intervals, in order to express semantics of vague-SWRL conveniently, we also employ intervals to represent single-valued weight. And omitting a weight is equivalent to specifying an interval [1, 1].

In what follows, we discuss the semantic of vague rule (1): a vague interpretation I satisfies $\bigwedge_{i=0...n} [vc_i * fdw_i]_{[vcv_i * sdw_i]} \wedge \bigwedge_{j=0...n} [vp_j * fdw_j]_{[vpv_j * sdw_j]} \rightarrow vc * w(or\ vp * w)$,

written $I \models \bigwedge_{i=0...n} [vc_i * fdw_i]_{[vcv_i * sdw_i]} \wedge \bigwedge_{j=0...n} [vp_j * fdw_j]_{[vpv_j * sdw_j]} \rightarrow vc * w\ (\ or\ vp * w\ )$,

if and only if $t\ [\underset{i=0,...n}{t}\ (f\,(fdw_i,\ sdw_i, vc_i)),\ \underset{j=0,...n}{t}\ (f\,(fdw_j,\ sdw_j, vp_i))] \le w_t(w, c(I))\cdot$

Where $f\ (fdw,sdw,d)$ is the function of first degree weight, second degree weight and vague membership degrees, $w_t(a,b)=\sup\{x|t(a,x)\le b\}$is R-implication, w represents the weight of atomic consequent, $c(I)$ stands for the interpretation of vague class or vague property, and f and w_t should satisfy equation: $w(t(a,w(a,b),b)=[1,1]$.

Based on RuleML 0.9 [3] and existing uncertainty extension for RuleML in [1], we extend RuelML with the first degree weight, second degree weight and membership degrees. We introduce <fdw>…</fdw> and <sdw>…</sdw> into RuleML to represent the first degree weight and the second degree weight, respectively. Also, <degree>…</degree> stands for the degrees of vague classes or properties. In addition, <weight>…</weight> represents the weight of atomic consequent.

3 Conclusion

Based on vague sets, this paper has proposed vague-SWRL, presented second degree weight to modify the membership degrees of vague classes and vague properties and extended RuleML to represent vague-SWRL. Vague-SWRL combining vague sets with SWRL suffices to employ membership degree intervals to represent fuzzy knowledge more accurately. What's more, second degree weight can balance and supplement first degree weight. And, vague-SWRL is an acceptable format for RIF.

References

1. Damásio, C.V., Pan, J.Z., Stoilos, G., Straccia, U.: Representing Uncertainty in RuleML Fundamenta Informaticae, vol. 82, pp. 1–24. IOS Press, Amsterdam (2008)
2. Gau, W.L., Buehrer, D.J.: Vague sets. IEEE Transactions on Systems, Man, and Cybernetics 23(2), 610–614 (1993)
3. Hirtle, D., Boley, H., Grosof, B., Kifer, M., Sintek, M., Tabet, S., Wagner, G.: Schema Specification of RuleML 0.9, `http://www.ruleml.org/0.9/`
4. Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language | Combining OWL and RuleML. W3C Member Submission, `http://www.w3.org/Submission/SWRL/`
5. Pan, J.Z., Stoilos, G., Stamou, G., Tzouvaras, V., Horrocks, I.: f-SWRL: A Fuzzy Extension of SWRL. Journal of Data Semantic, special issue on Emergent Semantics 4090, 28–46 (2006)
6. Zadeh, L.A.: Fuzzy sets. Information and Control 8(3), 338–353 (1965)
7. The Rule Markup Initiative, `http://www.ruleml.org/`

Alternative Strategies for Contextual Reasoning with Conflicts in Ambient Computing

Antonis Bikakis and Grigoris Antoniou

Institute of Computer Science, FO.R.T.H., Heraklion, Greece
{bikakis,antoniou}@ics.forth.gr

Abstract. Reasoning in Ambient Computing environments requires formal models that represent ambient agents as autonomous logic-based entities, and support sharing and distributed reasoning with the available ambiguous context information. This paper presents an approach from the field of Multi-Context Systems that handles these requirements by modeling contexts as local rule theories in a P2P system and mappings through which the agents exchange context information as defeasible rules, and by performing some type of distributed defeasible reasoning.

1 Motivation

The study of ambient computing environments has introduced new research challenges in the field of Distributed Artificial Intelligence. These are mainly caused by the imperfect nature of *context* and the special characteristics of the agents that provide and process this knowledge. Context may be *unknown*, *ambiguous*, *imprecise*, and *erroneous*. The ambient agents are expected to have different goals, experiences and perceptive capabilities and may use distinct vocabularies. Due to the highly dynamic and open nature of the environment, they are not able to know a priori all other entities that are present at a specific time instance nor can they communicate directly with all of them. So far, most pervasive computing frameworks have followed fully centralized approaches for managing and reasoning about context, which helped them to achieve better control, and better coordination between the participating entities. However, such solutions cannot meet the demanding requirements of ambient environments. The dynamics of the network and the unreliable and restricted wireless communications inevitably lead to fully distributed solutions.

2 Proposed Reasoning Approach

We propose a fully distributed approach for reasoning in ambient computing environments, which is based on the Multi-Context Systems (*MCS*) paradigm. Our approach models a MCS P as a collection of distributed local rule theories P_i in a P2P system: $P = \{P_i\}, i = 1, 2, ..., n$. Each system node (context) has a proper distinct vocabulary V_i and a unique identifier i. Each local theory is a set of rules that contain only local literals (literals from the local vocabulary).

These are of the form: $r_i^l : a_i^1, a_i^2, ...a_i^{n-1} \rightarrow a_i^n$ where i denotes the node identifier. These rules express local knowledge and are interpreted in the classical sense: whenever the premises of a local rule derive as logical consequences of the local theory, then so does the conclusion of the rule. Each node also defines mappings that associate its local literals with literals from the vocabulary of other nodes (*foreign literals*). The mappings are modeled as defeasible rules of the form: $r_i^m : a_i^1, a_j^2, ...a_k^{n-1} \Rightarrow a_i^n$. The above mapping rule is defined by P_i, and associates some of its own local literals with some of the local literals of P_j, P_k and other system nodes. Finally, each node P_i defines a trust level order T_i, which includes a subset of the system nodes, and expresses the trust that P_i has in the other system nodes. This is of the form: $T_i = [P_k, P_l, ..., P_n]$. A node P_k is considered more trusted by P_i than node P_l if P_k precedes P_l in this list.

Even if it is assumed that each node's local theory is consistent, this will not necessarily hold for the global knowledge base. The unification of the local context theories may result in inconsistencies caused by the mappings. Previous works on MCS either neglected this problem ([1,2]), or did not use any preference or priority mechanism to resolve the conflicts ([3]). Based on the model that we describe above, we propose four alternative strategies that use context and trust information from the system nodes to resolve potential conflicts. We have implemented the strategies in four versions of a Logic Programming algorithm for query evaluation. The *Single Answers* strategy requires each node to return only the truth value of the literal it is queried about. When a node receives two conflicting answers from two different nodes, it resolves the conflict by comparing the trust it has in the two nodes. The *Strength of Answers* strategy requires the queried node to return additional information about whether the computed answer derives from the local theory of the queried node or from its mappings. In the *Propagating Supportive Sets* strategy, a queried node also returns a set of node *ids* describing the most trusted (according to its own trust level ordering) *course of reasoning* that leads to the computed answer. The querying node computes its confidence in the returned answer based on the trust it has in the nodes contained in this set. Finally, in the *Complex Supportive Sets* strategy, the queried node returns all possible *courses of reasoning* that lead to the computed answer. The most trusted course is then determined by the node that issues the query. The analysis of the four algorithms reveals a tradeoff between the extent of context knowledge that each strategy exploits to resolve the potential conflicts, and the computational overhead that it imposes on the system nodes.

References

1. McCarthy, J., Buvač, S.: Formalizing Context (Expanded Notes). In: Aliseda, A., van Glabbeek, R., Westerståhl, D. (eds.) Computing Natural Language, pp. 13–50. CSLI Publications, Stanford (1998)
2. Ghidini, C., Giunchiglia, F.: Local Models Semantics, or contextual reasoning=locality+compatibility. Artificial Intelligence 127(2), 221–259 (2001)
3. Brewka, G., Roelofsen, F., Serafini, L.: Contextual Default Reasoning. IJCAI, 268–273 (2007)

Taming Existence in RDF Querying

François Bry[1], Tim Furche[1], Clemens Ley[2], Benedikt Linse[1], and Bruno Marnette[2]

[1] Institute for Informatics, University of Munich, Oettingenstraße 67, D-80538 München, Germany
[2] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, England

Abstract. We introduce the recursive, rule-based RDF query language RDFLog. RDFLog extends previous RDF query languages by arbitrary quantifier alternation: blank nodes may occur in the scope of all, some, or none of the universal variables of a rule. In addition RDFLog is aware of important RDF features such as the distinction between blank nodes, literals and URIs or the RDFS vocabulary. The semantics of RDFLog is closed (every answer is an RDF graph), but lifts RDF's restrictions on literal and blank node occurrences for intermediary data. We show how to define a sound and complete operational semantics that can be implemented using existing logic programming techniques. Using RDFLog we classify previous approaches to RDF querying along their support for blank node construction and show equivalence between languages with full quantifier alternation and languages with only $\forall\exists$ rules.

1 Introduction

With the staggering amount of data available in RDF form on the Web, the second indispensable ingredient becomes the easy selection and processing of RDF data. For that purpose, a large number of RDF query languages has been proposed. In this paper, we add a further exemplar: RDFLog extends datalog to support the distinguishing features of RDF such as blank nodes and the logical core [1] of the RDFS vocabulary. In RDFLog, Blank nodes can be constructed by existentially quantified variables in rule heads. RDFLog allows *full alternation* between existential and universal quantifiers in a rule. This sharply contrasts with previous approaches to rule-based query languages that either do not support blank nodes (in rule heads) at all [2], or only a limited form of quantifier alternation [3].

To illustrate the benefits of full quantifier alternation, imagine an information system about university courses. We distinguish three types of rules with existential quantifiers (and thus blank nodes) based on the alternation of universal and existential quantifiers:

(1) "Someone knows each professor" can be represented in RDFLog as

$$\exists stu \forall prof\ ((prof, \mathsf{rdf{:}type}, \mathsf{uni{:}professor}) \rightarrow (stu, \mathsf{foaf{:}knows}, prof)) \qquad (1)$$

We call such rules ∃∀ rules. Some approaches such as [5] are limited to rules of this form. We show that a recursive rule language that is limited to these kind of rules is strictly less expressive than a language that allows rules also of the form discussed under (2) and (3). The gain is that languages with only ∃∀ rules are still decidable. However, we also show that there are larger fragments of RDFLog that are still decidable.

(2) Imagine, that we would like to state that each lecture must be "practiced" by another course (such as a tutorial or practice lab) without knowing more about that course. This statement can not be expressed by ∃∀ rules. In RDFLog it can be represented as

$$\forall lec \exists crs \big((lec, \mathsf{rdf{:}type}, \mathsf{uni{:}lecture}) \rightarrow (crs, \mathsf{uni{:}practices}, lec) \big) \tag{2}$$

Such rules are referred to as ∀∃ rules. Recent proposals for rule extensions to SPARQL are limited to this form, if they consider blank nodes in rule heads at all. The reason is that in SPARQL `CONSTRUCT` patterns a fresh blank node is constructed for every binding of the universal variables.

(3) To the best of our knowledge, RDFLog is the first RDF query language that supports the third kind of rules, where quantifiers are allowed to alternate freely: This allows to express statements such as, for each lecture there is a course that "practices" that lecture and is attended by all students attending the lecture. This is represented in RDFLog as

$$\begin{aligned} &\forall lec \exists crs \forall stu \big((lec, \mathsf{rdf{:}type}, \mathsf{uni{:}lecture}) \wedge (stu, \mathsf{uni{:}attends}, lec) \rightarrow \\ &\qquad (crs, \mathsf{uni{:}practices}, lec) \wedge (stu, \mathsf{uni{:}attends}, crs) \big) \end{aligned} \tag{3}$$

We show (for the first time) that rules with full quantifier alternation can be normalized to ∀∃ form. Thus full quantifier alternation does not add to the expressiveness of RDFLog. Rather, for all languages with ∀∃ rules it comes by virtue of the rewriting presented here for free, despite being considerably more convenient for the programmer.

References

1. Muñoz, S., Pérez, J., Gutierrez, C.: Minimal deductive systems for RDF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519. Springer, Heidelberg (2007)
2. Polleres, A.: From SPARQL to rules (and back). In: Proc. Int'l. World Wide Web Conf (WWW). ACM Press, New York (2007)
3. Schenk, S., Staab, S.: Networked graphs: A declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the Web. In: Proc. Int'l. World Wide Web Conf (WWW). ACM Press, New York (2008)
4. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572. Springer, Heidelberg (2002)
5. Yang, G., Kifer, M.: Reasoning about anonymous resources and meta statements on the Semantic Web. Journal of Data Semantics 1 (2003)

A Tableau Algorithm for Possibilistic Description Logic $\mathcal{ALC}$[*]

Guilin Qi[1] and Jeff Z. Pan[2]

[1] Institute AIFB, University of Karlsruhe, Germany
gqi@aifb.uni-karlsruhe.de
[2] Department of Computing Science, The University of Aberdeen
jpan@csd.abdn.ac.uk

1 Introduction

Uncertainty reasoning and inconsistency handling are two important problems that often occur in the applications of the Semantic Web, such as the areas like medicine and biology [2]. Possibilistic description logics, first proposed by Hollunder in [1], are extension of description logics with possibilistic semantics. It is well-known that possibilistic logic is a powerful logical framework for dealing with uncertainty and handling inconsistency.

In this paper, we propose a tableau algorithm for possibilistic DL $\mathcal{ALC}$. We give tableau expansion rules for computing the inconsistency degree of a possibilistic $\mathcal{ALC}$-ABox. We show that our algorithm is sound and complete.

2 Tableau Algorithms for Inference in Possibilistic DL $\mathcal{ALC}$-ABoxes

In this section, we extend tableau expansion rules for DL $\mathcal{ALC}$ to possibilistic DL $\mathcal{ALC}$. Let $\mathcal{A}$ be a possibilistic $\mathcal{ALC}$-ABox. Without loss of generality, we assume that all the concepts appearing in $\mathcal{A}$ are in *negation normal form* (NNF).

To compute the inconsistency degree of $\mathcal{A}$, we construct a completion forest $\mathcal{F}_\mathcal{A}$ from $\mathcal{A}$. Each node x in the completion forest is labelled with a set of weighted concepts $\mathcal{L}(x)$ and each edge $\langle x, y\rangle$ is labelled with a set of weighted role names $\mathcal{L}(\langle x, y\rangle)$. If a weighted concept C^α is in $\mathcal{L}(x)$, it means that x belongs to concept C with necessity degree α. Similar comment is applied to weighted role names. The completion forest $\mathcal{F}_\mathcal{A}$ is initialized such that it contains a root node x_a, with $\mathcal{L}(x_a) = \{C^\alpha | ((C(a), \alpha) \in \mathcal{A}\}$, for each individual a occurring in $\mathcal{A}$, and an edge $\langle x_a, x_b\rangle$, with $\mathcal{L}(\langle x_a, x_b\rangle) = \{r^\alpha : (r(a, b), \alpha) \in \mathcal{A}\}$, for each pair (a, b) of individual names for which $\mathcal{L}(\langle x_a, x_b\rangle)$ is non-empty. We then apply the following expansion rules:

- $\sqcap$-rule: if
  - [1] $(C_1 \sqcap C_2)^\alpha \in \mathcal{L}(x)$, x is not blocked, and
  - [2] there are no $\beta \geq \alpha$ and $\gamma \geq \alpha$ such that $\{(C_1)^\beta, (C_2)^\gamma\} \subseteq \mathcal{L}(x)$
  - then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(C_1)^\alpha, (C_2)^\alpha\}$

[*] Guilin Qi is partially supported by the EU under the IST project NeOn and the X-Media project, and Jeff Z. Pan is partially supported by the EU MOST project.

- $\sqcup$-rule: if
  - [1] $(C_1 \sqcup C_2)^\alpha \in \mathcal{L}(x)$, x is not blocked, and
  - [2] there are no $\beta \geq \alpha$ and $\gamma \geq \alpha$ such that $\{(C_1)^\beta, (C_2)^\gamma\} \subseteq \mathcal{L}(x)$
  - then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C^\alpha\}$ for some $C \in \{C_1, C_2\}$
- $\exists$-rule: if
  - [1] $(\exists r.C)^\alpha \in \mathcal{L}(x)$, x is not blocked, and
  - [2] there is no y such that $r^\beta \in \mathcal{L}(\langle x, y\rangle)$ where $\beta \geq \alpha$ and $C^\gamma \in \mathcal{L}(y)$ where $\gamma \geq \alpha$,
  - then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{r^\alpha\}$ and $\mathcal{L}(y) = \{C^\alpha\}$
- $\forall$-rule: if
  - [1] $(\forall r.C)^\alpha \in \mathcal{L}(x)$, x is not blocked, and
  - [2] there is a y such that $r^\beta \in \mathcal{L}(\langle x, y\rangle)$ with $C^\gamma \notin \mathcal{L}(y)$ for $\gamma \geq min(\alpha, \beta)$,
  - then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C^{min(\alpha,\beta)}\}$

The tableau algorithm stops when it encounters a clash: a completion forest $\mathcal{F}$ in which $\{A^\alpha, (\neg A)^\beta\} \subseteq \mathcal{L}(x)$ for some node x and some concept name A, where $\alpha, \beta > 0$. In this case, the completion forest contains an inconsistency and the inconsistency degree, denoted $d_{Inc}(\mathcal{F})$, is $min(\alpha, \beta)$. If the algorithm stops and all of the forest i ($i = 1, ..., n$) contain an inconsistency with inconsistency degree α_i (in this case $\alpha_i > 0$), then the inconsistency degree of $\mathcal{A}$ is $min(\alpha_1, ..., \alpha_n)$. Otherwise, if the algorithm stops and there is a forest that does not contain an inconsistency, then the possibilistic $\mathcal{ALC}$-ABox is consistent.

In our algorithm, it is very critical to expand the weighted concept in a right order. That is, we apply tableau rules to expand those concepts with the highest weight first, then those concepts with the second highest weight, and so on. For each individual name in $\mathcal{A}$, the forest contains a root node, which will be called as *old nodes*. The nodes that are created by the $\exists$-rule are called *new nodes*. To ensure that our algorithm only requires space polynomial in $|\mathcal{A}|$, we apply tableau expansion rules in the following order:

- apply the $\sqcap$-rule and the $\sqcup$-rule to old nodes as long as possible and check for clash;
- treat each old node in turn, generate all the necessary direct successors of it in a depth first manner by applying the $\exists$-rule and the $\forall$-rule, and check for clash;
- successively handle the successors in the same way.

Given a complete forest $\mathcal{F}_\mathcal{A}$, we can transform it to a possibilistic $\mathcal{ALC}$-ABox $\mathcal{A}_\mathcal{F}$ as follows $\mathcal{A}_\mathcal{F} = \cup_{\mathcal{L}(x_a) \in \mathcal{F}_\mathcal{A}} \{(C(a), \alpha) : C^\alpha \in \mathcal{L}(x_a)\} \cup_{\mathcal{L}(\langle x_a, x_b\rangle) \in \mathcal{F}_\mathcal{A}} \{(r(a,b), \alpha) : r^\alpha \in \mathcal{L}(\langle x_a, x_b\rangle)\}$. Then $Inc(\mathcal{F}_\mathcal{A}) = Inc(\mathcal{A}_\mathcal{F})$ is the inconsistency degree of $\mathcal{F}_\mathcal{A}$.

Theorem 1. *(Soundness and Completeness) Let $\mathcal{M} = \{\mathcal{F}_\mathcal{A}^1, ..., \mathcal{F}_\mathcal{A}^n\}$ be a set of complete forests constructed from $\mathcal{A}$ by application of tableau expansion rules. Suppose $\mathcal{M}'$ is a set of complete forests obtained from $\mathcal{M}$ by application of a tableau expansion rule. If $Inc(\mathcal{M}) = \alpha$ then $Inc(\mathcal{M}') = \alpha$. Let $d_{Inc}(\mathcal{F}_\mathcal{A}^i) = \alpha_i$, then we have $d_{Inc}(\mathcal{M}) = Inc(\mathcal{M})$, where $d_{Inc}(\mathcal{M}) = min\{\alpha_1, ..., \alpha_n\}$.*

References

1. Hollunder, B.: An alternative proof method for possibilistic logic and its application to terminological logics. In: Proc. of UAI 1994, pp. 327–335 (1994)
2. Udrea, O., Deng, Y., Hung, E., Subrahmanian, V.S.: Probabilistic ontologies and relational databases. In: Proc. of CoopIS/DOA/ODBASE 2005, pp. 1–17 (2005)

Author Index