



LarKC

*The Large Knowledge Collider
a platform for large scale integrated reasoning and Web-search*

FP7 – 215535

D1.1.3

Initial Knowledge Representation Formalism

**Florian Fischer
Uwe Keller
Atanas Kiryakov
Zhisheng Huang
Vassil Momtchev
Elena Simperl
Dieter Fensel
Dumitru Roman**

Document Identifier:	LarKC/2008/D1.1.3/Vx.x
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	version 1.1
Date:	September 3, 2009
State:	final
Distribution:	public

EXECUTIVE SUMMARY

This document describes a recommended knowledge representation formalism that meets the requirements of LarKC. More specifically it provides an analysis of requirements stemming from the use cases in the project and surveys the landscape of existing knowledge representation languages on the Semantic Web. Furthermore it examines systems that are able to perform inference at a large scale. Based on these results it proceeds to define its two main contributions: (i) A recommended language called *L2* (short for **LarKC Language**) that goes slightly beyond RDFS by including tractable parts of OWL and (ii) A framework that is used for the formal specification of *L2* and can be used for the specification of tractable languages within the established Semantic Web language stack by providing the used vocabulary along with a set of entailment rules. This framework can in turn also be used to establish additional custom extensions of the language.

L2 was determined based on both theoretical complexity results and practicability of its implementation, i.e. in existing inference systems, and allows two slightly different semantics according to the specific requirements in a scenario. It is balanced around two goals: To guarantee a baseline of interoperability between different scenarios by means of a common language and by being open ended and lightweight enough to allow for use-case specific extensions within the defined framework, and by this means to support the open nature of the LarKC platform. As such it is a tractable minimal language that supports the overall vision of LarKC.



DOCUMENT INFORMATION

IST Project Number	FP7 – 215535	Acronym	LarKC
Full Title	Large Knowledge Collider		
Project URL	http://www.larkc.eu/		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	Number	1.1.3	Title	Initial Knowledge Representation Formalism
Work Package	Number	1	Title	Conceptual Framework & Evaluation

Date of Delivery	Contractual	M7	Actual	31-Oct-08
Status	version 1.1		final	<input type="checkbox"/>
Nature	prototype <input type="checkbox"/> report <input checked="" type="checkbox"/> dissemination <input type="checkbox"/>			
Dissemination Level	public <input type="checkbox"/> consortium <input checked="" type="checkbox"/>			






Authors (Partner)	Florian Fischer (UIBK), Uwe Keller (UIBK), Elena Simperl (UIBK), Atanas Kiryakov (Ontotext), Vassil Momtchev (Ontotext), Zhisheng Huang (VUA), Dieter Fensel (UIBK), Dumitru Roman (UIBK)			
Resp. Author	Florian Fischer (UIBK)		E-mail	florian.fischer@sti2.at
	Partner	UIBK	Phone	+43 (512) 507 6488

Abstract (for dissemination)	<p>This document describes a recommended knowledge representation formalism that meets the requirements of LarKC. More specifically it provides an analysis of requirements stemming from the use cases in the project and surveys the landscape of existing knowledge representation languages on the Semantic Web. Furthermore it examines systems that are able to perform inference at a large scale. Based on these results it proceeds to define its two main contributions: (i) A recommended language called <i>L2</i> (short for LarKC Language) that goes slightly beyond RDFS by including tractable parts of OWL and (ii) A framework that is used for the formal specification of <i>L2</i> and be used for the specification of tractable languages within the established Semantic Web language stack by providing the used vocabulary along with a set of entailment rules. This framework can in turn also be used to establish additional custom extensions of the language. <i>L2</i> was determined based on both theoretical complexity results and practicability of its implementation, i.e. in existing inference systems, and allows two slightly different semantics according to the specific requirements in a scenario. It is balanced around two goals: To guarantee a baseline of interoperability between different scenarios by means of a common language and by being open ended and lightweight enough to allow for use-case specific extensions within the defined framework, and by this means to support the open nature of the LarKC platform. As such it is a tractable minimal language that supports the overall vision of LarKC.</p>
Keywords	ontology reasoning, Web scale reasoning



Version Log			
Issue Date	Rev No.	Author	Change
June 16, 2008	1	Florian Fischer	Creation
June 30, 2008	2	Florian Fischer	Added Introduction
August 10, 2008	3	Florian Fischer	Added OWL overview
September 9, 2008	4	Zhisheng Huang	Added RIF overview
September 9, 2008	5	Uwe Keller	Completed the WSML overview
September 9, 2008	6	Zhisheng Huang	Added Common Logic overview
September 15, 2008	7	Zhisheng Huang	Updated Section 3.5 and Section 3.6
October 24, 2008	8	Uwe Keller, Florian Fischer	Finalization of the document
November 19, 2008	9	Florian Fischer	Started rewrite
November 22, 2008	10	Florian Fischer	Restructuring Completed

PROJECT CONSORTIUM INFORMATION

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI 	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB http://www.astrazeneca.com/	ASTRAZENECA 	Bosse Andersson AstraZeneca Lund, Sweden E-mail: bo.h.andersson@astrazeneca.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL 	Emanuele Della Valle CEFRIEL SCRL. Milano, Italy E-mail: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O. http://cyceurope.com/	CYCORP 	Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia E-mail: witbrock@cyc.com
Hchstleistungsrechenzentrum, Universitaet Stuttgart http://www.hlrs.de/	HLRS 	Dr. Georgina Gallizo Hchstleistungsrechenzentrum, Universitaet Stuttgart Stuttgart, Germany E-mail : gallizo@hlrs.de
Max-Planck-Institut fr Bildungsforschung http://www.mpib-berlin.mpg.de/index_js.en.htm	MAXPLANCKGESELLSCHAFT 	Michael Schooler, Max-Planck-Institut fr Bildungsforschung Berlin, Germany E-mail: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO 	Atanas Kiryakov, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: atanas.kiryakov@sirma.bg
SALTLUX INC. http://www.saltlux.com/EN/main.asp	Saltlux 	Kono Kim SALTLUX INC Seoul, Korea E-mail: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT http://www.siemens.de/	Siemens 	Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT Muenchen, Germany E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD http://www.shef.ac.uk/	Sheffield 	Prof. Dr. Hamish Cunningham THE UNIVERSITY OF SHEFFIELD Sheffield, UK E-mail: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM http://www.vu.nl/	Amsterdam 	Prof. Dr. Frank van Harmelen VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands E-mail: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY http://www.iwici.org/	WICI 	Prof. Dr. Ning Zhong THE INTERNATIONAL WIC INSTITUTE Mabeshi, Japan E-mail: zhong@maebashi-it.ac.jp




<p>INTERNATIONAL AGENCY FOR RESEARCH ON CANCER http://www.iarc.fr/</p>	<p>IARC2</p>  The logo of the International Agency for Research on Cancer (IARC) is a blue emblem. It features a central caduceus (a staff with two snakes entwined around it) superimposed on a map of the world. The entire emblem is encircled by a laurel wreath.	<p>Dr. Paul Brennan INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France E-mail: brennan@iarc.fr</p>
--	--	---

TABLE OF CONTENTS

1	INTRODUCTION	1
2	REQUIREMENTS AND MOTIVATION FOR A KNOWLEDGE REPRESENTATION FORMALISM IN LARKC	3
2.1	General Requirements	3
2.2	Requirements from Concrete Use-Case Scenarios	4
2.2.1	WP6 Urban Computing	4
2.2.2	WP7 Early Clinical Drug Development and Carcinogenesis Research	5
2.3	Summary of Requirements	6
2.3.1	Design Goals	6
2.3.2	Objectives	6
3	STATE-OF-THE-ART IN ONTOLOGY LANGUAGES	8
3.1	Resource Description Framework, and RDF(S)	8
3.2	OWL	9
3.3	OWL 2	10
3.3.1	OWL 2 Profiles	10
3.4	The WSML Language	11
3.4.1	WSML Language Variants	12
3.5	RIF - Rule Interchange Format	14
3.5.1	RIF as a Rule Language for the Semantic Web	14
3.5.2	RIF Logic Dialects	15
3.5.3	Relevance for LarKC	15
3.6	Common Logic	16
4	LANGUAGE SUPPORT IN SCALABLE INFERENCE ENGINES	18
5	LARKC KNOWLEDGE REPRESENTATION FORMALISM	21
5.1	Language Features and Informal Semantics of <i>L2</i>	21
5.2	Formal Semantics of <i>L2</i>	24
6	CONCLUSIONS AND FUTURE WORK	31
A	APPENDIX I	32
A.1	Datamodel	32
B	APPENDIX II	35
B.1	Basic Definitions	35
B.2	Entailment and Consistency Checking Rules	36
B.3	Semantics for Entailment and Consistency Checking Rules	40

1 INTRODUCTION

The objective of the project LarKC is to build a generic, highly extendable platform for distributed incomplete reasoning that can scale to the dimensions of the current and future Web. From a theoretical perspective this will be achieved by defining a fundamentally novel approach to reasoning. Traditional Semantic Web reasoning methods will be enriched with and complemented by methods from adjacent areas of Computer Science and beyond which have proven to be successful in processing, managing and storing massive amounts of data or information: Information Retrieval, Machine Learning, Cognition, Database Management Systems, to name only a few. From an technological perspective, LarKC will result into an open platform, which can be extended through easily combinable plug-ins for exposing various methods, techniques and heuristics from the areas previously mentioned. A series of core plug-ins will be developed within the project as part of the core technical work or in the context of the three use cases. A considerably larger number of plug-ins is envisioned to be developed, however, by the external parties, academics as well as industry, which operate in the knowledge technologies sector. Plug-ins are then arranged in particular, concrete pipelines for a specific scenario, which can then be executed within LarKC. This execution in turn is controlled by a special kind of plug-in, called a “decider”.

Plug-in developers are expected to design each of these steps in various ways. With LarKC the developers will be provided with a generic, robust platform, deployed over a computing cluster and via “computing at home”, which can be used to experiment with these plug-ins and to build knowledge-based applications. In addition, in order to support the realization of LarKC plug-ins beyond the boundaries of the project consortium, and to ensure the further development of the platform within a sustainable community of developers worldwide LarKC defines an integration framework complemented by a solid engineering methodology, standard APIs and knowledge representation formalisms.

The integration framework and methodology aim at assisting the users of the LarKC platform and the developers of LarKC-supported knowledge-based applications in the usage of the available technological components and the realization of custom plug-ins. Standard APIs, the open, generic architectural design and the different forms of deployment of the platform provide the core technological requirements for the realization of such applications. The LarKC knowledge representation formalism, the subject of this deliverable and its extended version (to be delivered later in the project), is expected to contribute to the generally applicable character of the overall platform and associated technological components. By design it should allow for the accommodation of a wide range of domain and application requirements referring to the representation of knowledge and reasoning thereupon, and should provide a basis for the development of reasoning plug-ins to be used in such applications.

Thus the result of this deliverable and its follow-up version have to be balanced between two design goals. The minimal LarKC knowledge representation formalism should guarantee a certain interoperability between different plug-ins by providing a common representation language, and on the other hand be flexible enough as to not limit the inherent extensibility of the platform. Enforcing a comprehensive new formalisms on all potential plug-ins and applications built on top of LarKC technology contradicts the primary aim of the project to realize an open, generic distributed Semantic Web reasoning platform. Instead, this deliverable defines a recommended

baseline language that should be supported by platform components developed in example in work packages WP2, WP3 and WP5. In summary, this language needs to i) be aligned with existing standards, ii) support the overall goals of the LarKC platform in terms of an open and extensible platform, iii) enable tractable large scale reasoning.

This deliverable provides a first version of this language which is built upon a careful analysis of the use case requirements complemented by a case requirements complemented by a literature survey of some of the most popular knowledge representation formalisms that have emerged in the last decades of research in the relevant communities. The follow up version of this deliverable will be concerned with the evaluation and revision of this language, as well as possible extensions that are even more tailored towards very specific key application scenarios of LarKC technology. Furthermore the final version of this deliverable will be realigned with the outcomes of work currently conducted in the OWL working group towards OWL 2, a revised version of OWL.

The rest of this document is structured as follows. Chapter 2 describes general, as well as use-case-driven requirements for the LarKC knowledge representation formalism. Chapter 3 provides an overview of the KR languages that are expected to be relevant for Web scale reasoning tasks to be run on the LarKC platform, while Chapter 4 provides an empirical look at language features that are currently being implemented practically in inference engines that scale to very large datasets. Chapter 5 presents the current version of the language. Finally, Chapter 6 concludes this deliverable with a summary of its contributions and an outline of future works.

2 REQUIREMENTS AND MOTIVATION FOR A KNOWLEDGE REPRESENTATION FORMALISM IN LARKC

2.1 General Requirements

The LarKC project aims to go beyond current solutions for Semantic Web reasoning in three crucial aspects.

1. Scalability: By building a massive distributed, parallel and incomplete reasoning platform.
2. Heterogeneity: By going beyond current logic-based reasoning methods and employing methods from information retrieval, machine learning, information theory, probabilistic reasoning, . . .
3. Reusability: By engineering the LarKC platform as a pluggable and extensible architecture, in order to invite researchers to create own components and run them on the LarKC platform.

In more detail, the LarKC platform is envisioned to employ probabilistic and approximate methods for selection, abstraction (machine learning, inductive inference) and reasoning (deductive inference) to achieve its goals. Thus approximate and incomplete reasoning is one of the main approaches to perform inference at a Web scale. However there are two further aspects that need to be taken into consideration for a knowledge representation (KR) formalism in this regard.

First of all it has to be in line with the overall vision of LarKC in terms of scalability and at the same time provide a suitable level of expressivity for practical applications. More precisely it should support the greatest possible degree of expressivity while still being a tractable language, i.e. it should have polynomial (data complexity). As such it should move beyond pure RDF and RDF Schema but still be a suitable extension of them and not ignore existing standards – ideally the LarKC knowledge representation formalism should also be aligned with existing standardization efforts layered on top of RDF.

The second aspect supporting this notion is that the Large Knowledge Collider is supposed to serve as an experimental platform for other researchers and support a number of heterogeneous plug-ins. As such (1) it cannot ignore existing Web standards, (2) should provide a baseline formalism that needs to be supported by all plugins, but should be extensible along the line of more expressive standard languages *and* fit in existing frameworks. This is crucial to realize the vision of an open platform that does not tie future extensions to one specific formalism.

In terms of complexity several common reasoning tasks need to be considered for a KR formalism in the LarKC platform. For a more in-depth review of use-cases and tasks for Web scale reasoning we refer to D4.1. In this deliverable we limit ourselves to the following four reasoning tasks, as in propositionally closed languages different reasoning tasks can usually be reduced to one another in polynomial time.

- Query answering
- Checking for the consistency of an ontology
- Checking satisfiability of a concept

- Testing for concept subsumption
- Checking for the concept membership of an instance

Moreover “complexity” refers to the overall complexity, composed of data complexity measured in relation to the number of facts, as well as taxonomic complexity in regard to the number of axioms. The rationale is that both, the number of facts and the number of axioms will be very large in knowledge bases at a Web scale. However, it is clear, that the number of facts within a Web scale KB will most often be significantly bigger than the schema-level (terminological) axioms.

Following this general requirements for a suitable knowledge representation formalism in the scope of the LarkC project we now examine individual use-cases. The purpose of this is to clarify (1) if requirements from them match and support the general expectations towards a KR formalism, and (2) if additional requirements can be identified which can practically fulfilled.

2.2 Requirements from Concrete Use-Case Scenarios

2.2.1 WP6 Urban Computing

The first use-case has very clear requirements in regard to knowledge representation within LarkC. The main issue in this use-case from the perspective of a KR formalism is the vast heterogeneity of data. Heterogeneity can be interpreted as representational (syntax), semantic or in terms of applied underlying default assumptions.

Representative Heterogeneity refers to the issue that data can be equally described in different specification languages (surface syntaxes), which however share underlying semantics, possibly in the sense that the two formalisms are in a representational subsumption relation. This requirement stems from the fact that urban computing data is available from a multitude of different source and needs to be collected and aggregated. These sources deliver data in a variety of different formats, ranging from relational DBMS to more expressive semantic languages (RDF(S), OWL, WSMML). From a thematic point of view this data includes traffic data, data about events (which includes spatial and temporal data), , and so forth. The integration and reuse of those data sets, therefore, need a process of conversion and translation for the data to become useful together.

Semantic Heterogeneity describes the requirement to support different kinds of underlying formal systems and thus also for multiple paradigms of reasoners. For instance, many applications of Urban Computing may need different reasoners for temporal reasoning, spatial reasoning (e.g. supporting the Region Connection Calculus [48]), and causal reasoning.

As practical example of semantic heterogeneity, consider that existing Description Logic reasoners are not able to perform topological reasoning generally. Spatial reasoning often involves topological relationships, determined by means of geometrical calculations based on instance data. Description Logic reasoners are often not able to compute the required topology in this way, thus, they are limited to reasoning over an associated ontology and can provide additional

qualitative background information but not work quantitatively on the instance data itself (see [22] for more background).

Albeit it is not necessary to accommodate all these existing formalisms in one KR language, it is however required to take the possibility for various extensions into account and define a way to integrate them. Moreover, an additional requirement is to describe and select components in the system on the basis of available data and the desired processing tasks.

Default Heterogeneity means that different default assumptions can be applied in the context of a specific formalism. Concrete examples include the Closed World Assumption (CWA), Open World Assumption (OWA), Unique Name Assumption (UNA) and Non-unique Name Assumption. While in a Semantic Web setting the OWA and the Non-unique Name Assumption are the generally adopted paradigms, in specific applications it is often convenient to apply a local CWA, i.e. time tables, city maps, . . . can all be assumed to be complete. In such cases not being able to find a public transport between two points at a certain time likely means that there does not exist one. Similar concerns apply in the case of the UNA, as streets and bus stops clearly have unique names.

Thus a formalism should not set down these assumptions in stone and provide the possibility to include components and data based on different semantic defaults.

2.2.2 WP7 Early Clinical Drug Development and Carcinogenesis Research

Both use-cases in WP7 make use of large-scale RDF based knowledge bases combined from various data sources, forming a “Linked Life Data” repository. The initial story boards of both use-cases involve light-weight reasoning on RDF data sources and to a certain extent as well DL-like reasoning for the detection of inconsistencies. Further requirements result from the various data-sources that are supposed to be integrated.

PubMed¹ provides online access to the major life science bibliographic database, MEDLINE. PubMed Central (PMC) gives free online access to the full text of a subset of PubMed articles, plus some additional material. The estimated size of semantically annotate all of the abstracts in PubMed, or at least MEDLINE would amount to 5 billion RDF triples. Annotating entire texts at the token level would increase this estimate to over 20 billion triples². Including relevant biomedical ontologies and knowledge bases such as UMLS, GO (The Gene Ontology), HapMap, Entrez gene database, Uniprot and IARC lung cancer data sets increases this size even more. Combined these datasets are expected to exceed one billion RDF triples as well.

Thus a clean layering on top of RDF is mandatory for this use case. Moreover this use-cases enforce the requirement of a lightweight and tractable formalism due to the sheer size of initial data sources. Moreover formal specifications of the semantics of these sources exist in wildly varying degrees – ranging from OWL ontologies (i.e. for Uniprot RDF) to simple RDBMS schemas (MEDLINE). Thus different degrees of expressivity at the schema level need to be supported as well.

¹<http://www.ncbi.nlm.nih.gov/pubmed/>

²For more details on these estimates please see D7b.1.1b “Requirements summary and data repository”

2.3 Summary of Requirements

Based on general requirements towards a KR formalism and the requirements resulting from the use-cases we now define (1) a set of high level goals and (2) a set of definite requirements for knowledge representation in LarKC. In the following, we outline the plan of how to satisfy these requirements.

2.3.1 Design Goals

Tractability of the formalism As motivated by the general scalability aims and the requirements for light-weight reasoning over very large data repositories in the use cases, the KR formalism has to be inherently tractable.

Expressivity of the formalism At the same time language expressivity should clearly go beyond the modeling primitives in RDF(S). The motivation is that a language with too limited expressivity will provide too few opportunities in practical use-cases, i.e. the inability express inconsistencies and thus also to detect them.

Consistency with existing Web standards The LarKC knowledge representation formalism must not ignore existing web standards. This includes clean layering on top of RDF, the possibility to align with RDFS and OWL, as well as XML and XML Schema. Furthermore the standard principles such as using URIs as identifiers for concepts, ontologies, ... should be adopted.

Easy of integration of existing ontologies and ontology interoperability As motivated from the use-cases, it should be reasonably easy to integrate different existing knowledge bases from distinct providers. This demands primitives that for example allow to express or derive the equality between individuals or concepts in domain models to be integrated.

2.3.2 Objectives

From the set of high-level design goals in the previous section we now derive Objectives for the LarKC knowledge representation formalism.

Complexity of the Formalism Inference in the formalism must be tractable for large datasets and compare favorably to existing standards.

Concept definitions On the epistemological level, the language has to be able to form complex definitions of classes as far as this is in line with scalability requirements.

Property definitions The language has to be able to form definitions of properties as far as this is in line with scalability requirements.

Data type support The language has to provide a set of standard data types, based on XML schema data-types.

Equivalence The language has to provide support to state equivalence between concepts as well as individuals. This is crucial in the scope of integration of existing ontologies and ontology interoperability.

Meta-modeling The language has to support a certain degree of meta-modeling, i.e. to treat concepts as instances and vice-versa. The reason for that is again interoperability between existing ontologies, which might differ in the separation between concepts and instances.

Annotation support The language should provide support the possibility to further annotate statements. Rationals for this include (but are not limited to) the requirement to add timestamps or confidence levels to statements, as motivated in the use-cases.

Identification of ontologies and concepts URIs must be used to identify concepts as well as complete ontologies. The motivation for this is a) consistency with existing Web standards, b) integration and interoperability with existing ontologies.

Support for meta-data It should be possible to augment an ontology with meta-data like author, version, . . . in an extensible fashion.

3 STATE-OF-THE-ART IN ONTOLOGY LANGUAGES

3.1 Resource Description Framework, and RDF(S)

Several standardized languages together form the base layer of the Semantic Web layer cake. At the very bottom of this layered architecture resides the Resource Description Framework (RDF)[38]. RDF is a formalism to represent knowledge about resources on the Web. In that sense a resource can be an arbitrary anything (object) that can be identified by a URI and spoken about on the Web.

RDF is a comparatively basic assertional language, that can be used to state facts about resources and that uses a triple syntax for that purpose. As such assertions are of the following basic form `< subject, predicate, object >`. Predicates can be understood as the attributes of a specific resource, and thus form attribute-value pairs. A statement like $p(x, y)$ can be understood to assign object y as a value for attribute p for resource x . Arguments to predicates in an RDF statement must always be ground, with the exception being introduced due to *anonymous objects/blank nodes* which can be represented by local existential variables. A statement such as `friend(Johnny, _1), loves(_1, Carmen)` asserts that Jonny has an unknown friend who is known to love Carmen. Based on that very simple conceptual model it is clear a set of triples can also be interpreted as an *RDF graph* where subjects and objects are considered as nodes and the predicates connecting them form the edges.

The semantics of RDF (and also RDF(S), see below) is defined in [30] based on the formal concept of *entailment* on RDF graphs. This formal specification of the semantics is roughly based on and akin to the model theory in [31]. An RDF expression A is said to entail another RDF expression B if every possible interpretations that makes A true also makes B true. There are different so called *entailment regimes* layered upon each other, which diverge in the kind of inferences they allow. More precisely, an entailment regime is a binary relation between subsets of RDF graphs and RDF graphs, which is transitive and idempotent. Entailment regimes are tied to specific RDF “vocabularies” or rather to a specific interpretation of such an vocabulary that assigns special meaning to the symbols in the vocabulary. Therefore interpretations are also named accordingly to a particular vocabulary, i.e. *rdf-interpretation*, *rdfs-interpretation*.

An interpretation without any special vocabulary attached, that is one which does not pay any attention to names of nodes in the graph, is also called *simple interpretation* from which *simple entailment*, the most basic form of RDF entailment follows: A graph S entails a graph E if and only if a subgraph of S is an instance of E, where an *instance* of some graph is the graph with its blank nodes replaced and a *subgraph* of a graph is a subset of its triples.

The next entailment regime satisfying additional semantic conditions in relation to a special vocabulary is *rdf-entailment*. Thus RDF interpretations define a set of additional semantic conditions on its vocabulary and furthermore also adds a set of *axiomatic triples* as predefined assumptions to the knowledge base. Most of the RDF axiomatic triples denote that anything can be regarded as a property, i.e. `< rdf:predicate, rdf:type, rdf:property >`. Entailment itself is actually defined in the same way as for simple entailment, apart from the important fact that it refers to RDF-entailment instead of simple entailment.

RDF Schema [8] goes beyond the basic modelling elements allowed by RDF by allowing to further describe (1) predicates and (2) the relationships between predicates and other resources. RDF Schema specifies a vocabulary to express classes (unary relations) and properties (binary relations), which can be ordered in a hierarchy for classes and a hierarchy of properties. Furthermore RDF(S) allows to specify domain and range for properties, that is it allows to type properties. Based on this it is possible to infer class membership and subclass relations, both by means of the subclass hierarchy, and furthermore class membership because of typed properties, property values and subproperty relations through the subproperty hierarchy. These extensions lead to yet another entailment regime and interpretation, based on the extended vocabulary and additional semantic conditions on them as well as an increased set of axiomatic triples.

A notable feature missing from RDF and as well from RDF(S) is *negation* or *disjunction*, which severely limits the expressive power of the language. This limited has two important consequences:

1. Key inference tasks for RDF Schema are still tractable (if blank nodes are used with care).
2. It is not possible to state inconsistencies in RDF (besides inconsistencies stemming from the wrong use of datatype values).

But although RDF and its different entailment regimes are conceptually very simple languages they are already expressive enough for many practical use-cases, as can be seen in Section 2.2. Moreover it clearly shows a syntactically limited language can be extended semantically by strengthening its semantic conditions with well defined modelling elements.

3.2 OWL

Description Logics [4] are used in a wide range of applications. Perhaps the most prominent application of DLs is, however, as the basis for ontology languages such as OIL, DAML+OIL [19] and OWL [42].

OWL is a semantic web ontology language, developed by the W3C Web-Ontology working group, in order to represent information about classes of objects and how objects are related, and additionally also information about objects themselves. The semantics of OWL can be defined via a mapping to a specific expressive Description Logic, however, OWL is also influenced by several other knowledge representation paradigms as for example its abstract syntax is very similar to frame-based KR approaches.

In fact OWL consists of three sublanguages, namely OWL-Lite, OWL-DL and OWL-Full, from which the first two have a model theory that is based on the already mentioned mapping to Description Logics and also only a restricted extension of the RDF semantics. OWL-Full's semantics is given in terms of an extension of the (slightly non-standard) RDF model theory. In turn every kind of OWL ontology is a valid RDF ontology, and every RDF ontology is an OWL Full ontology. However, there are cases where an RDF ontology will not be legal OWL-Lite or OWL-DL.

More precisely OWL-Lite corresponds to the Description Logic $\mathcal{SHIF}(D)$ while OWL-DL is equivalent to $\mathcal{SHOIN}(D)$. These DL-based language variants can be regarded as a TBox along with a role hierarchy (in Description Logic terms) that can

be used to describe a domain as concepts (classes) and roles (properties). Thus they significantly go beyond the simple ability of asserting facts such as RDF and the primitive class hierarchy and role hierarchy building blocks of RDF Schema. OWL concepts can be defined in terms of intersections, unions or complements of other concepts, and so just as in Description Logics it is possible to build more complex concepts from atomic concepts and properties using several constructors, depending on the language variant. Furthermore it is possible to denote subsumption relations between concepts and roles, equivalence, disjointness of concepts and the non-equivalence of individual objects. In regard to roles it is possible to assert that a role is functional, symmetric, transitive or inverse functional.

The close correspondence of OWL to Description Logics is by design in order to reuse available research results from Description Logic, in regard to complexity, reasoning techniques and readily available reasoner implementations for OWL. Furthermore, based on the mapping to Description Logics it is possible to cleanly layer different language fragments of varying exclusivity based on the syntactic constructs allowed in them. For example OWL-Lite's main purpose has been to build simple classification hierarchies with basic constraints (cardinality constraints with the value 0 or 1). The next iteration of OWL, OWL 2¹ continues in this fashion by specifying further language fragments for specific purposes.

3.3 OWL 2

OWL 2 [43] extends OWL with several new features which include extra syntactic sugar, property and qualified cardinality constructors, extended datatype support, puning as a simple form of meta-modelling, and furthermore different handling of annotations (they have no semantic meaning anymore).

OWL 2 DL is an extensions of OWL-DL and corresponds to the Description Logic *SR₀IQ*. However, OWL 2 is not given a semantics by mapping to *SR₀IQ* (since puning not being available in it) but rather directly on the constructs of the functional-style syntax for OWL 2. Similarly to OWL, OWL 2 again comes in several different flavors (now called *profiles* now): OWL 2 Full (which is an extension of the RDFS semantics again), OWL 2 DL, OWL 2 EL, OWL 2 QL, and OWL 2 RL, of which we briefly describe the three last ones, and the motivation to include them, in more detail below. Additionally OWL-DL could in theory be considered an OWL 2 profile as well.

3.3.1 OWL 2 Profiles

OWL 2 profiles in general can be regarded as language fragments with special desirable computational properties that are tailed for specific applications. Most of the profiles are defined by enforcing syntactic restrictions and thus also limiting the expressivity of OWL 2 – in that sense they all trade expressive power for efficient reasoning.

OWL 2 EL, which is based on $\mathcal{EL}++$ (see [3] and [2]) is a fragment that is very tractable in regard to several key inference tasks such as consistency checking, subsumption, instance checking or classification – they can be decided in polynomial time. On the other hand it is still expressive enough to adequately model many real world problems. The most prominent constructs from OWL 2 that are disallowed in

¹At the time of the writing this document a W3C Working Draft.

OWL 2 EL are disjunction, negation, enumerations of multiple elements, inverse and irreflexive object properties, functional object properties, symmetric and asymmetric object properties as well as several constructs involving universal quantification.

OWL 2 QL is based on DL-Lite [17] (which is actually not a single language fragment but rather a family of languages with several slight variations) and tailored for reasoning over large instance sets combined with a relatively inexpressive TBox. More specifically, many important reasoning tasks can be performed in logarithmic space with respect to the size of the ABox. The variant picked up in OWL 2 is DL-Lite_R and supports property inclusion axioms. Other variants instead support (inverse) functionality on object properties.

A notable feature goes hand in hand with DL-Lite's complexity results: Since the data complexity of query answering over knowledge bases is within the AC⁰. Moreover, it is possible to separate TBox and ABox reasoning for the evaluation of a query. Based on this complexity result it is in turn possible to delegate the ABox reasoning to a standard SQL database engine. Thus, OWL 2 QL is optimized for data complexity and supports very efficient query answering even for large instance sets. A detailed examination of complexity results for the DL-Lite family of languages is available within [1].

The OWL 2 RL is a fragment which is customized to support reasoning with rule-based engines by only considering objects that are explicitly named in the knowledge base. It is profile that is intended to form a proper extensions of RDFS while still being computationally tractable. As such it realizes a weakened form of the OWL 2 Full semantics and is very similar in spirit to DLP [28] and pD* entailment [51]. The OWL 2 RL semantics are provided as an partial axiomatization in the form of additional entailment rules directly on the RDF serialization of OWL 2.

3.4 The WSML Language

The Web Service Modeling Language WSML [53] is a concrete formal language based on the conceptual model of WSMO [24]. WSML is a formal language to describe various elements of a service-oriented system semantically. As such, it provides a language for describing ontologies, goals, Web services, mediators, and their interrelationships in the way envisioned by WSMO. Besides providing an ontology language for use with Web service description, WSML also allows using the Semantic Web ontology languages RDF schema [9] and OWL (DL) [21] for describing the terminologies used in goal and Web service descriptions.

The semantic foundation of any description in WSML is the *ontology language* part of WSML which is used for describing the terminology. WSML recognizes two important Knowledge Representation paradigms in this context, namely Description Logics [5] and logical rule-based languages [40]. The user may choose which paradigm to use: Description Logics, rules, a common subset, or a common superset. To this end, WSML defines a number of different *variants*: WSML-Core marks the common subset, WSML-DL marks the Description Logics paradigm, WSML-Rule marks the rules paradigm, and WSML-Full marks the common superset.

WSML defines an ontology language for WSML-Full. The other variants are obtained by suitably restricting the syntax of the language. The language variant also determines which Semantic Web ontology languages may be used. WSML-Core and Rule permit the use of a subset of OWL DL, inspired by [29]. The DL and Full variants

permit the use of arbitrary OWL DL. Finally, a subset of RDF Schema may be used with WSML-Core and DL, and all of RDF Schema may be used with WSML-Rule and Full.

The WSML ontology language can be seen as a sub-language of WSML. The other sub-languages are the languages used for the functional, nonfunctional, and behavioral description, i.e., the languages used to realize WSMO capabilities, non-functional properties, and choreographies. These languages all allow using terminology defined in ontologies, and there is considerable overlap between the ontology languages, and specifically language of logical expressions, and the expressions used in these other sub-languages, as we shall see in this chapter.

Orthogonal to the 4 sub-languages are the three concrete syntaxes of WSML for the specification and exchange of WSML descriptions. The surface syntax is the primary syntax of WSML; its structure and keywords are based on the WSMO conceptual model, and it is primarily meant for the specification and viewing of WSML descriptions by human users. The XML and RDF representations are meant for the exchange of WSML descriptions over the Web, as well as RDF-based access of WSML descriptions.

3.4.1 WSML Language Variants

Following the principle, detailed in the previous section, of using both the Description Logic (DL) and Logic Programming² (LP) paradigms, WSML incorporates a number of different language *variants*, corresponding to the DL and LP paradigms, and their possible (subset and superset) interaction. Figure 3.1 shows the WSML language variants and their interrelationships. The variants differ in logical expressiveness and underlying language paradigms; they allow users to make a trade-off between the expressiveness of a variant and the complexity of reasoning for ontology modeling on a per-application basis. Additionally, different variants are more suitable for different kinds of reasoning and representation.

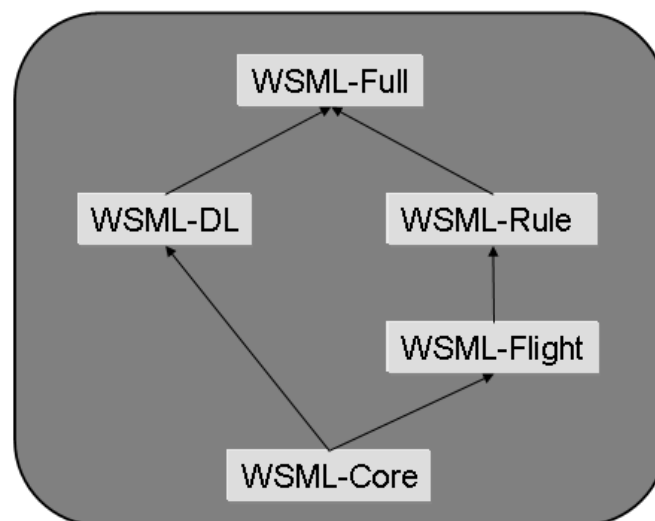


Figure 3.1: WSML Variants

²With Logic Programming we mean a declarative logic-based rules language with negation under the Well-Founded [25] or the Stable Model [26] Semantics.

WSML-Core is based on an intersection of the Description Logic \mathcal{SHIQ} and Horn Logic, also known as Description Logic Programs [29]. It has the least expressive power of the WSML variants, and is a common subset of the DL-based and LP-based variants. That is, every WSML-Core description is also a WSML-DL and WSML-Flight/Rule description.

WSML Core ontologies, goals, and Web services may import OWL DLP (a subset of OWL DL) ontologies.

WSML-DL is the Description Logic variant of WSML, and captures the Description Logic $\mathcal{SHIQ}(\mathbf{D})$, which corresponds to a large part of (the DL species of) OWL [21]. Furthermore, this variant is used for the interoperation with OWL DL ontologies. WSML-DL ontologies, goals, and Web services may import OWL DL ontologies.

WSML-Flight is the least expressive of the two LP-based variants of WSML. Compared with WSML-Core, it adds features such as meta-modeling, constraints, and nonmonotonic negation. WSML-Flight is based on a Logic Programming variant of F-Logic [37] and is semantically equivalent to Datalog with inequality and (locally) stratified negation [47].

Technical issues related to the layering between the (DL-based) WSML-Core and (F-Logic-based) WSML-Flight are discussed in detail in [15], and the specific layering issues in WSML are discussed in detail in [13].

Finally, the WSML-Flight variant can be used for combinations with RDF graphs and RDFS ontologies. WSML-Flight ontologies, goals, and Web services may import RDF graphs and RDFS ontologies.

WSML-Rule extends WSML-Flight with further features from Logic Programming, namely the use of function symbols, unsafe rules, and unstratified negation. There are two prominent semantics for logic programs with unstratified negation, namely the Stable Model Semantics [26] and the Well-Founded Semantics [25]; with respect to the task of query answering, the latter can be seen as an approximation of the former. Since the Stable Model Semantics is more general, WSML-Rule (in version 0.3 [10]) adopts the Stable Model Semantics, but implementations may use the Well-Founded Semantics as an approximation for the task of query answering.

WSML-Full unifies WSML-DL and WSML-Rule. A definition of the semantics for WSML-Full, generalizing WSML-DL and WSML-Rule, is proposed in [13, 14]. However, as the combination of Description Logics and nonmonotonic logic programs is still an open research issue – some work has been done on the topic [23, 49, 11, 12, 44, 45, 16], but there is no consensus on which is the right semantics for the combination. Therefore, the current version of WSML does not define a semantics for WSML-Full, but instead outlines a number of properties such a semantics should have.

WSML has two alternative layerings, namely, $\text{WSML-Core} \Rightarrow \text{WSML-DL} \Rightarrow \text{WSML-Full}$ and $\text{WSML-Core} \Rightarrow \text{WSML-Flight} \Rightarrow \text{WSML-Rule} \Rightarrow \text{WSML-Full}$. For both layerings, WSML-Core and WSML-Full mark the least and most expressive layers. The two layerings are to a certain extent disjoint, namely the inter-operation between

WSML-DL, on the one hand, and WSML-Flight and -Rule, on the other, is only possible through a common subset (WSML-Core) or through a very expressive superset (WSML-Full). The precise properties of language layering in WSML are discussed in [20].

3.5 RIF - Rule Interchange Format

RIF (Rule Interchange Format) is a rule language, which is developed by the Rule Interchange Format (RIF) Working Group of W3C³. The basic goal of the Rule Interchange Format (RIF) Working Group is to devise such standards and make sure that they are not only useful in the current environment, but are easily extensible in order to deal with the evolution of rule technology and other enabling technologies.

3.5.1 RIF as a Rule Language for the Semantic Web

RIF is designed to a standard rule language for the Semantic Web. RIF is intended to be a W3C specification that builds on and develops the existing range of specifications that have been developed by the W3C, such as RDF/RDFS and OWL. RDF data and RDFS and OWL ontologies are represented using RDF graphs. RIF does not provide a format for exchanging RDF graphs; it is assumed that RDF graphs are exchanged using RDF/XML, or any other syntax that can be used for representing or exchanging RDF graphs⁴.

As reported in the document on RIF RDF and OWL Compatibility⁵, a typical scenario for the use of RIF with RDF/OWL is the exchange of rules that use RDF data and/or RDFS or OWL ontologies: an interchange partner A has a rules language that is RDF/OWL-aware, i.e., it supports the use of RDF data, it uses an RDFS or OWL ontology, or it extends RDF(S)/OWL. A sends its rules using RIF, possibly with references to the appropriate RDF graph(s), to partner B. B receives the rules and retrieves the referenced RDF graph(s) (published as, e.g., RDF/XML). The rules are translated to the internal rules language of B and are processed, together with the RDF graphs, using the RDF/OWL-aware rule engine of B. The use case Vocabulary Mapping for Data Integration is an example of the interchange of RIF rules that use RDF data and RDFS ontologies.

A RIF document that refers to (imports) RDF graphs and/or RDFS/OWL ontologies, or any use of a RIF document with RDF graphs, is viewed as a combination of a document and a number of graphs and ontologies. RIF provides a mechanism for referring to (importing) RDF graphs and a means for specifying the context which corresponds to the intended entailment regime in RDF/RDFS. A RIF-OWL-combination consists of a RIF document and a set of RDF graphs, analogous to a RIF-RDF combination, because the syntax for exchanging OWL ontologies is based on RDF graphs.

There is a correspondence between statements in RDF graphs and certain kinds of formulas in RIF. Namely, there is a correspondence between RDF triples of the form $\langle s, p, o \rangle$ and RIF frame formulas of the form $s'[p' \rightarrow o']$, where s' , p' , and o' are RIF symbols corresponding to the RDF symbols s , p , and o , respectively. This means that

³<http://www.w3.org/2005/rules/wg>

⁴<http://www.w3.org/2005/rules/wiki/SWC>

⁵<http://www.w3.org/2005/rules/wiki/SWC>

whenever a triple $\langle s, p, o \rangle$ is satisfied, the corresponding RIF frame formula $s'[p' \rightarrow o']$ is satisfied, and vice versa.

The following example illustrates the interaction between RDF and RIF:

The following RDF statements

```
ex:john ex:brotherOf ex:jack.
```

```
ex:jack ex:parentOf ex:mary.
```

state that ex:john is a brother of ex:jack and ex:jack is a parent of ex:mary.

A RIF rule

```
forall ?x ?y ?z (?x[ex:uncleOf -> ?z] :-  
  And(?x[ex:brotherOf -> ?y] ?y[ex:parentOf -> ?z]))
```

says that whenever some x is a brother of some y and y is a parent of some z, then x is an uncle of z. From this combination the RIF frame formula :john[:uncleOf -> :mary], as well as the RDF triple :john :uncleOf :mary, can be derived.

3.5.2 RIF Logic Dialects

RIF consists of a family of RIF dialects. Each dialect is a collection of components that works together, forming an interlingua. New dialects are needed when no existing dialect provides the required rule-language features for interchange.

The RIF Framework for Logic-based Dialects (RIF-FLD)⁶ describes mechanisms for specifying the syntax and semantics of logic-based RIF dialects through a number of generic concepts. Currently, the first two existing RIF dialects are the RIF Basic Logic Dialect (RIF-BLD)⁷ and the RIF Production Rules Dialect (RIF-PRD)⁸.

- RIF-BLD (Basic Logic Dialect) is a specialization of RIF-FLD capable of representing definite Horn rules with equality enhanced with a number of syntactic extensions to support expressive features such as objects and frames, internationalized resource identifiers (IRIs) as identifiers for concepts, and XML Schema data types.
- RIF-PRD (Production Rules Dialect) specifies a production rules dialect to enable the interchange of production rules.

3.5.3 Relevance for LarKC

In this subsection, we will briefly discuss the relevance of RIF for LarKC. We will examine RIF with respect to the requirements which are analyzed in Section 2.3, i.e., i) Tractibility of the formalism, ii) Expressivity of the formalism, iii) Consistency with existing Web standards, and iv) Easy of integration of existing ontologies and ontology interoperability.

Tractibility of the formalism. RIF is designed to a standard rule language for the Semantic Web. Thus, we would not expect an easy solution for its tractibility, like that of other rule languages such as SWRL⁹.

⁶<http://www.w3.org/2005/rules/wiki/FLD>

⁷<http://www.w3.org/2005/rules/wiki/BLD>

⁸<http://www.w3.org/2005/rules/wiki/PRD>

⁹<http://www.w3.org/Submission/SWRL/>

Expressivity of the formalism. RIF consists of a family of RIF dialects. Its expressivity is strong.

Consistency with existing Web standards. RIF documents can refer to (imports) RDF graphs and/or RDFS/OWL ontologies. It supports the existing web standards very well. Furthermore, it is well consistent with the existing Web Standard.

Easy of integration of existing ontologies and ontology interoperability. Similar answer with the previous one. Namely, RIF is easy to integrate existing ontologies like RDF/RDFS/OWL. However, it is still an open question whether or not RIF can be easily integrated different existing knowledge bases from distinct providers. We have not yet seen any work which report that RIF can support Default Heterogeneity and other heterogeneities which are analyzed in Chapter 2.

3.6 Common Logic

Common logic (CL) is a formal language based on first-order logic, intended to facilitate the exchange and transmission of knowledge in computer-based systems. Common Logic is developed by the Common Logic Working Group¹⁰, an ISO effort for an international standard of logic-based languages.

The main goals of the Common Logic Working are to develop a framework for a family of logic-based languages which have the following features:

- With a semantics that is a superset of the semantics of many other logic-based languages.
- With an abstract syntax that can be specialized to the concrete syntaxes of other logic-based languages.
- Designed to preserve the semantics when information is interchanged among heterogeneous systems.

The purpose of those features is to guarantee that content exchanged between CL-conformant languages has the same semantics in each language.

CL is a family of first-order logics which share a common abstract syntax and model theory, and an XML framework for encoding and transmitting them, or their content, on an open network. Thus, each CL-conformant concrete syntax adds grammar rules to express the abstract syntax in concrete symbols. Three concrete syntaxes included in the CL document: i) KIF: Knowledge Interchange Format¹¹, ii) CGIF: Conceptual Graph Interchange Format¹², and iii) CLML: Common Logic Markup Language (XML based)¹³. Every CL feature has a concrete expression in KIF, CGIF, and CLML. Any statement in any other CL-conformant language can be translated to KIF, CGIF, or CLML while preserving the original semantics.

¹⁰<http://cl.tamu.edu/>

¹¹<http://www-ksl.stanford.edu/knowledge-sharing/kif/>

¹²http://www.jfsowa.com/cg/cgstandw.htm#Header_44

¹³<http://www.jfsowa.com/talks/clprop.htm>

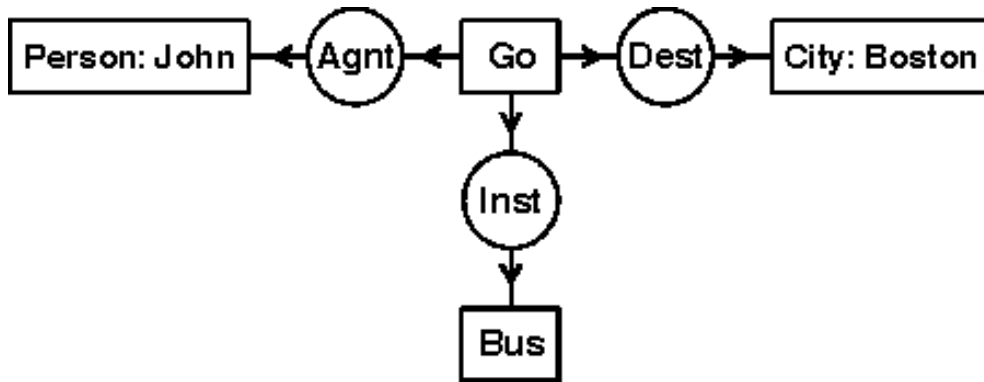


Figure 3.2: "John is going to Boston by bus."

CGIF is a Conceptual Graph Interchange Format, which is developed based on conceptual graphs (CGs)¹⁴. CGs have been developed as a conceptual schema language, as specified by ISO/IEC 14481 on Conceptual Schema Modeling Facilities (CSMF). CGIF has been designed for interchange between CSMF systems or between other IT systems that require a structured representation for logic. The CG Standard provides guidance for implementers of IT systems that use conceptual graphs as an internal representation or as an external representation for interchange with other IT systems. The external representations are readable by humans and may also be used in communications between humans or between humans and machines.

Figure 3.2 shows a CG which displays the form for "John is going to Boston by bus." The CG in this figure can be represented by using CGIF as follows:

[Go *x] (Agnt ?x [Person: John]) (Dest ?x [City: Boston]) (Inst ?x [Bus])

The semantic web languages, such as RDF/RDFS and OWL, can be mapped into Common Logic. A RDF triple $\langle s, p, o \rangle$ can be translated into its corresponding CL formula as $(p \ s \ o)$.

Common Logic is in fact a family of first-order logics which share a common abstract syntax and model theory, including the associated high complexity and expressivity. As such it is not a tractable formalism that is immediately relevant in LarKC, although it provides an easy way to convert RDF/RDFS/OWL data into CL formulas in the sense that these languages are only regarded as a specific CL "dialect". In this way, Common Logic is easy to integrate existing ontologies and ontology interoperability. However, again, Common Logic does not support the specific heterogeneities which are analyzed in Chapter 2.

¹⁴<http://www.jfsowa.com/cg/cgstandw.htm>

4 LANGUAGE SUPPORT IN SCALABLE INFERENCE ENGINES

LarKC is meant to be a platform for web-scale reasoning and therefore the recommended KR language for LarKC needs to take relevant Web standards into account and adopt them where appropriate. The Web has a number of standards for object identification and the representation and manipulation of data which can be directly adopted by any Web language.

Although not embodying the only source of scalability in LarKC, the recommended KR formalism must not disregard the inherent trade-off between complexity of reasoning and language expressivity, which is for example outlined in [7]. This jump from a tractable to an effectively intractable language, often called “computational cliff” usually happens in very distinct steps caused by specific combinations of modelling primitives. However, an informed decision is not possible based only on such purely theoretical considerations but also has to take practical user requirements towards the expressivity of the language into account.

In order to ensure that both the desired level of expressivity and practical feasibility of the KR formalism are in line with LarKC’s requirements (see Chapter 2), we initially examined existing large-scale, RDF-based inference engines regarding the respective (OWL, RDFS, ...) vocabularies and inference levels supported. These results are useful to indicate what modelling primitives are useful *beyond* those that are obvious choices due to known complexity results, based on user requirements towards them. Along with existing complexity results, this does not only regard language features from their purely theoretical side but it also gives an impression based on:

1. The relevance of specific language constructs for users.
2. The practicability of implementing certain language features efficiently.

Thus, we compared different levels of vocabulary supported in OWLIM¹, Oracle 11g² and AllegroGraph RDFStore³.

AllegroGraph RDFStore is a disk-storage based, persistent RDF graph database with support for SPARQL and Prolog rules. Its inference capabilities cover RDFS and a select subset of OWL as well. AllegroGraph also goes beyond the basic RDF data-model by supporting quints. A first three slots are used as subject, predicate, object by AllegroGraph, as usual. The two remaining slots are used to support named-graphs, which can be used for arbitrary attributes such as weights, trust factors, times, ..., and a further internal identifier which is used for administrative purposes but can also be referred to by other triples. This approach solves issues that are a result of traditional reification (“triple bloat”).

Oracle 11g provides support for semantic technologies in Oracle’s database products. It can be used as RDF store and inference engine that supports various “rule-bases”, which represent different sets of entailment rules: RDFS++ (RDFS plus `owl:sameAs` and `owl:InverseFunctionalProperty`), OWLSIF (based on the pD*

¹<http://www.ontotext.com/owlim/index.html>

²http://www.oracle.com/technology/tech/semantic_technologies/index.html

³<http://agraph.franz.com/allegrograph/>

vocabulary in [35] and supporting intensional “if-semantics” in place of stronger “iff-semantics” for OWL constructs) and OWLPrime [54]. OWLPrime is the most comprehensive of the three rule sets, consisting of over 50 distinct entailment rules. By providing support for named graphs (quads) Oracle 11g goes beyond the pure triple data model of RDF as well.

OWLIM supports different sets of entailment rules in a similar fashion: OWL-Max, OWL-Horst, and RDFS. RDFS provides the usual RDFS semantics as defined in [32] but skips parts of D-entailment due to performance reasons. OWL-Horst (the default rule set) again corresponds roughly to the set of pD* entailment rules from [35]. OWL-Max, as the most comprehensive semantics in OWLIM, is close to OWL-Horst but goes slightly beyond it (basically its expressiveness covers OWL Lite combined with unrestricted RDFS). In regard to the data model supported OWLIM goes beyond pure RDF as well and supports a “rich RDF model” which includes i) named graphs and ii) so called triple sets⁴.

The respectively supported OWL subsets are summarized in Table 4.1 to illustrate the deltas that exist between them. Trivial constructs are omitted in the table. Primitives which are only partially supported are explicitly marked so. For example the standard OWL entailments for `owl:someValuesFrom` are often only supported in one direction (“if-semantics”). As can be seen, the respectively supported subsets are often very close to each other.

⁴<http://www.ontotext.com/ordi/index.html>

<i>Modelling Primitive</i>	<i>RDFS</i>	<i>OWL-Horst</i>	<i>OWL-MAX</i>	<i>OWLPrime</i>	<i>OWLSIF</i>	<i>RDFS++ (Oracle)</i>	<i>RDFS++ (Allegro)</i>
rdfs:domain	y	y	y	y	y	y	y
rdfs:range	y	y	y	y	y	y	y
rdfs:subClassOf	y	y	y	y	y	y	y
rdfs:subPropertyOf	y	y	y	y	y	y	y
Datatype Maps	y	y	y	y	y	y	
owl:AllDifferentFrom			p				
owl:FunctionalProperty		y	y	y	y		
owl:InverseFunctionalProperty		y	y	y	y	y	
owl:SymmetricProperty		y	y	y	y		
owl:TransitiveProperty		y	y	y	y		y
owl:allValuesFrom		y	y	p	p		
owl:cardinality			p				
owl:complementOf				p			
owl:differentFrom		y	y	y			
owl:disjointWith		y	y	y			
owl:equivalentClass		y	y	y	y		
owl:equivalentProperty		y	y		y		
owl:hasValue		y	y	p	p		y
owl:intersectionOf			y				
owl:InverseOf		y	y	y	y		y
owl:maxCardinality			p				
owl:minCardinality			p				
owl:oneOf			p				
owl:sameAs		y	y	y	y	y	y
owl:someValuesFrom		y	y	p	p		
owl:unionOf			p				

Table 4.1: Modelling Primitives

5 LARKC KNOWLEDGE REPRESENTATION FORMALISM

5.1 Language Features and Informal Semantics of *L2*

Based on theoretical complexity results as well as considerations based on the results of analysing which language primitives are typically supported in very scalable inference engines in practice, this section now defines the LarkC KR language, henceforth called *L2* (“LarkC Language”). For this purpose we select the supported language primitives, which mostly consist of the RDFS vocabulary and a limited sub-set of OWL that is still inherently tractable and allows extensions towards rule-based formalisms as well as towards Description Logics. In this way we aim to come up with a lightweight ontology language that serves as recommended language in LarkC.

Of the upcoming OWL 2 profiles, *L2* is closest to OWL 2 RL in spirit, as its formal semantics are also provided by means of a (partial) axiomatization in the form of entailment rules, that largely reflects a subset of the OWL 2 RDF based semantics¹ (see Appendix‘B). However, it deliberately further limits allowed language elements in order to allow clean future extensions towards ELP [39], which can be regarded to subsume the OWL 2 EL and OWL 2 QL profiles, but still has polynomial combined complexity and can polynomially transformed to Datalog. Thus *L2* can serve as a baseline for further extensions towards Description Logics as well as towards implementations on common rule engines.

L2 is layered on top of the datamodel explained in Appendix A.1 in the sense that it assigns a semantic interpretation to the first three elements of a triplesets, whereas further elements in the triplesets serve to modularize a dataset in a convenient way but do not have an immediate consequence on the semantic interpretation of the language.

As *L2* could be considered an OWL fragment (or “profile”) in the sense that it allows an efficient sub-set of inferences to be made, we can i.e. use the OWL 2 functional-style syntax² as high-level syntax. However, obviously any surface syntax with an appropriate mapping to the underlying RDF primitives can be used (e.g. [33]) as the fundamental design aspects of the language are independent of the particular syntax employed. This in turn greatly enhances the user friendliness of *L2* and facilitates the reuse of existing tools.

Supported language primitives

We will now enumerate the features in *L2* and explain why they are included and provide an informal descriptions of their intended semantics.

Class definitions (`rdfs:Class`) A class defines a set of individuals that belong together because they share common properties. We only allow partial class definitions in the language but not complete class definitions as they allow to emulate several other language features that are not explicitly included, i.e. class intersection.

Subclass descriptions (`rdfs:subClassOf`) *L2* allows to define class hierarchies in the same way as already RDFS does. Thus the intended meaning is exactly the same: If the class C_1 is defined to be the subclass of a class C_2 then the set of

¹<http://www.w3.org/TR/owl2-rdf-based-semantics/>

²Draft available at: <http://www.w3.org/TR/owl2-syntax/>

individuals that “belong to” (are in the class extension of) C_1 should be a subset of those that belong to C_2 . Furthermore subclass relations are transitive and a class is a subclass of itself.

Property definitions (`rdf:Property`) Properties can be used to state specific relations, either between individuals or between individuals and plain data values.

Subproperty descriptions (`rdfs:subPropertyOf`) In the same fashion as for classes it is also possible to organize properties in hierarchies by stating that a property is a subproperty of a number of other properties. Just as `rdfs:supClassOf`, `rdfs:subPropertyOf` is transitive.

Domain and Range restrictions (`rdfs:domain` and `rdfs:range`) The domain and respectively the range of a property can be limited to objects of a certain class with the restriction that `owl:Nothing` cannot be used. The domain of a property restricts to what individuals the property is applicable, while the range restricts the set of values that a property can take. Both domain and range restrictions impose *global* restrictions on a property in this case – the restrictions apply independently to which specific class a property is applied.

It needs to be noted that for both domain and ranges it is actually possible to give two different kinds of interpretations, namely inferring and constraining. For example, assume an individual x which is related to another individual y via a certain property p , with a class C_1 as domain and another class C_2 as range. Then, applying an inferring interpretation, it is possible to conclude that x belongs to C_1 and furthermore that y belongs to C_2 . A constraining interpretation on the other hand would actually *check* that the individual is of the correct type, as a condition, and otherwise raise this as an error.

Both kind of semantics for domain and ranges are valid and a choice should be made depending on the requirements of an application.

Class equivalence (`owl:equivalentClass`) Two classes may be stated to be equivalent in $L2$ in which case they also have the same set of instances and moreover also share common super and subclasses. This functionality is useful to perform basic schema mapping. Class equivalence is obviously a symmetric, reflexive, and transitive property. Furthermore, class equivalence between two classes C_1 and C_2 does simply require two implications stating that C_1 is a subclass of C_2 and vice versa. In this sense it is cleanly layered on top of RDFS, where this functionality has already been available but there simply was no explicit syntax for it.

Transitive properties (`owl:TransitiveProperty`) Transitivity of properties can be expressed in $L2$, as it is in fact also present implicitly and required through various other language constructs. Transitivity has the usual meaning that a property p holds for a pair of individuals (x, y) and another pair (y, z) , then it also holds for (x, z) . Moreover, transitivity in $L2$ can even be used in an unrestricted form compared to OWL, since $L2$ does not include general cardinality restrictions, which easily lead to undecidability when combined naively with transitivity, as shown in [34].

Symmetric properties (`owl:SymetricProperty`) A symmetric property is a property that is true in both directions. *L2* allows for the specification of symmetric properties with the usual meaning; if a property p holds for a pair (x, y) , then it also holds for (y, x) .

Inverse properties (`owl:inverseOf`) Furthermore, properties can be stated to be the inverse of another property, i.e. `hasParent` and `hasChild`. If p_1 is the inverse of p_2 and an individual x is related to another individual y by p_1 , then y is related by p_2 to x .

Property equivalence (`owl:equivalentProperty`) Two properties may be stated to be equivalent in the same fashion as classes can. Equivalent properties relate one individual to the identical set of other individuals.

Individual equivalence (`owl:sameAs`) Individual equality is mainly included in the language for practical purposes as two distinct URIs can very well point to the same resource. While individual equality slightly raises the computational complexity (see [52] for an in-depth treatment) it can still be dealt with in practical implementations by various means.

Omitted language features

We deliberately omitted a set of language constructs in *L2*, most notably functional and inverse functional properties, universal value restrictions, existential value restrictions, and minimal and maximal cardinality constraints. These decisions are based on results, available i.e. in [6]. Functional properties (equivalent to a maximal cardinality constraint of 1) essentially allow derivation of equality between individuals, which (i) can happen in a very non-intuitive manner and (ii) also comes at a performance cost.

Universal value restrictions are not admitted as they can i.e. cause disjunctions with more than one positive literal when translated to a rule engine. As such it would require Disjunctive Logic Programming (as shown in [36]), which is already a computationally more expensive formalism to deal with. In the same turn existential value restrictions would be difficult to realize on a rule engine, as this would require to eliminate them through skolemization which requires support for function symbols.

Complexity considerations

In general we consider several relevant reasoning tasks as outlined in Section 2.1:

- Query answering
- Checking for the consistency of an ontology
- Checking satisfiability of a concept
- Testing for concept subsumption
- Checking for the concept membership of an instance

Due to its close relationship with pD^* [50] and OWL 2 RL known complexity and tractability carry over to *L2*, as it can be regarded as a subset. Moreover, as we specify the semantics for *L2* via a set of entailment rules, which are in fact Horn

rules. Thus we can ensure tractability by restricting the general format of specifying entailment rules. Based on that we can establish an upper bound of the computational complexity and tractability, similarly as in [50]. See Section B and Appendix B for an in-depth treatment.

5.2 Formal Semantics of L2

In this section we first define an minimal, extensible minimal representation language that addresses the desiderata above: strong alignment with Semantic Web standards, extensibility, minimal semantic commitment, and tractability:

- *Alignment with fundamental Semantic Web principles:* It has to follow the principles underlying widely used Semantic Web standard, in particular RDF and its underlying semantic model, the extensibility of the formalism by means of application-specific vocabulary and corresponding refinement of the basic semantic model.

The proposed formalism embodies the fundamental conception underlying RDF [30, 38] to view all knowledge on the Semantic Web as a labeled graph structure that is syntactically represented by a (huge) triple set.

- *Extensibility and minimal semantic commitment:*

It should be possible to gradually extend a language with constructs from various more expressive knowledge representation languages in a principled and controlled way.

For an extension towards richer knowledge representation languages, i.e. a particular language profile, additional modeling vocabulary can be defined. To capture the intended meaning of the added modeling vocabulary, the semantics need to be extended by adding the required semantic constraints concerning the additionally introduced modeling primitives.

To ensure minimal semantic commitment and achieve controlled extensions, we adopt the approach from [50] for extending RDFS in a user-defined, but controlled manner: on top of the basic data model, we add use entailment rules, which can be used to characterize the basic inferences which are possible (or better: which can be expected) in the enriched KR formalism.

A profile definition consists therefore of an identifier for the profile (e.g. a URI), a specification of the vocabulary which is added to the formalism, and the set of entailment rules which captures the intended semantics of the vocabulary introduced by the profile.

In principle, it is also possible to define different semantics that cover the same vocabulary but use different entailment rules for this vocabulary. Language profiles can extend existing semantics by introducing additional vocabulary and respective entailment rules.

- *Tractability:* By restricting the general format for specifying entailment rules in a profile as in [50], we can give a characterization of the computational complexity and tractability of the resulting profile. By sticking to particular constraints on

the entailment rules, it can be ensured to end up with a tractable formalism independent of what the entailment rules specify concretely.

The concept of language extension that we consider here is in line with the extension mechanism of RDF [30], i.e. the enrichment of vocabulary and the definition of corresponding augmented vocabulary interpretations and entailment regimes. The specification of the semantics of vocabulary by entailment rules is inspired by the RDF specification and the work on R -entailment in [50], or as well the OWL 2 RL profile.

Further, we add a new type of rules, consistency checking rules, which allow to represent database-style consistency checks. The upper bounds on the complexity of inference that have been shown for R -entailment carry over to the modified framework presented here.

For the precise basic definitions and technical details used subsequently we refer to Appendix B. Based on these, we now formally define a vocabulary and two slightly different semantics for $L2$. In general our approach is similar to the layering of OWL dialects [42] and the definition of special language profiles that is ongoing in the OWL2 working group [43]. It allows to provide language extensions that trade expressive power in regard to certain constructs for performance in regard to certain reasoning tasks. As such the main motivations for the definition of a distinct extension are:

- User requirements for more expressive, additional syntactic modelling primitives beyond those provided by the recommended language.
- Different user requirements with respect to semantic assumptions taken, a different “modelling style”, i.e. Open/Closed World Assumption, different treatment of domain and range constraints, Unique Name assumption, . . .
- Different computational characteristics concerning key reasoning tasks.

The precise semantics of $L2$ slightly extend RDFS towards OWL. It limits RDFS in the sense that it suppresses a specific set of trivial inferences for performance reasons (by disregarding the RDF and RDFS axiomatic triples which are depicted in Figure 5.1 and 5.2) and on the other hand extends the vocabulary to include support for common modelling primitives from OWL. It is an open option to add the complete set of axiomatic triples for RDF and RDFS for language extensions if desired, however this comes with an obvious performance penalty.

The set of admit-able inferences is specified as a subset of rules Rules_{L2} from the more comprehensive list of entailment rules in Figure 5.2 and Figure 5.1. From these rules we deliberately skip rules related to data-type reasoning and trivial inferences. They are included for the sake of completeness since they can form the basis for future extensions.

The basis of $L2$ vocabulary consists of the RDF and RDFS vocabularies:

Definition 5.2.1 (RDF Vocabulary) $V_{RDF} = \{ \text{rdf:type}, \text{rdf:Property}, \text{rdf:XMLLiteral}, \text{rdf:nil}, \text{rdf:List}, \text{rdf:Statement}, \text{rdf:subject}, \text{rdf:predicate}, \text{rdf:object}, \text{rdf:first}, \text{rdf:rest}, \text{rdf:Seq}, \text{rdf:Bag}, \text{rdf:Alt}, \text{rdf:value} \}$

Definition 5.2.2 (RDFS Vocabulary) $V_{RDFS} = \{ \text{rdfs:domain}, \text{rdfs:range}, \text{rdfs:Resource}, \text{rdfs:Literal}, \text{rdfs:Datatype}, \text{rdfs:Class}, \text{rdfs:subClassOf}, \text{rdfs:subPropertyOf}, \text{rdfs:member}, \text{rdfs:Container},$

	If G contains	where	then add to G
rdfp1	p type FunctionalProperty $u p v$		
rdfp2	p type Inverse-FunctionalProperty $u p w$ $v p w$	$v \in U \cup B$	v sameAs w u sameAs v
rdfp3	p type SymmetricProperty $v p w$	$w \in U \cup B$	$w p v$
rdfp4	p type TransitiveProperty $u p v$ $v p w$		$u p w$
rdfp5a	v sameAs w		v sameAs v
rdfp5b	$v p w$	$w \in U \cup B$	w sameAs w
rdfp6	v sameAs w	$w \in U \cup B$	w sameAs v
rdfp7	u sameAs v v sameAs w		u sameAs w
rdfp8ax	p inverseOf q $v p w$	$w, q \in U \cup B$	$w q v$
rdfp8bx	p inverseOf q $v q w$	$w \in U \cup B$	$w p v$
rdfp9	v type Class v sameAs w		v subClassOf w
rdfp10	p type Property p sameAs q		p subPropertyOf q
rdfp11	$u p v$ u sameAs u' v sameAs v'	$u' \in U \cup B$	$u' p v'$
rdfp12a	v equivalentClass w	$w \in U \cup B$	v subClassOf w
rdfp12b	v equivalentClass w	$w \in U \cup B$	w subClassOf v
rdfp12c	v subClassOf w w subClassOf v		v equivalentClass w
rdfp13a	v equivalentProperty w	$w \in U \cup B$	v subPropertyOf w
rdfp13b	v equivalentProperty w	$w \in U \cup B$	w subPropertyOf v
rdfp13c	v subPropertyOf w w subPropertyOf v		v equivalentProperty w
rdfp14a	v hasValue w v onProperty p $u p w$		u type v
rdfp14bx	v hasValue w v onProperty p u type v	$p \in U \cup B$	$u p w$
rdfp15	v someValuesFrom w v onProperty p $u p x$		
rdfp16	x type w v allValuesFrom w v onProperty p u type v $u p x$	$x \in U \cup B$	x type w

Figure 5.1: Complete rule set for P-entailment [50] with the subset considered in $L2$ marked.

```

rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
rdf:nil rdf:type rdf:List .
    
```

Table 5.1: RDF axiomatic triples

	If G contains	where	then add to G
lg	vpl	$l \in L$	vpb_l
gl	vpb_l	$l \in L$	vpl
rdf1	vpw		p type Property
rdf2-D	vpl	$l = (s, a) \in L_D^+$	b_l type a
rdfs1	vpl	$l \in L_p$	b_l type Literal
rdfs2	p domain u		v type u
rdfs3	p range u		v type u
rdfs4a	vpw	$w \in U \cup B$	w type u
rdfs4b	vpw	$w \in U \cup B$	w type Resource
rdfs5	v subPropertyOf w w subPropertyOf u		v subPropertyOf u
rdfs6	v type Property		v subPropertyOf v
rdfs7x	p subPropertyOf q		vqw
rdfs8	v type Class	$q \in U \cup B$	v subclassOf Resource
rdfs9	v subclassOf w u type v		u type w
rdfs10	v type Class		v subclassOf v
rdfs11	v subclassOf w w subclassOf u		v subclassOf u
rdfs12	v type Container-MembershipProperty		v subPropertyOf member
rdfs13	v type Datatype		v subclassOf Literal

Figure 5.2: Complete rule set for D-entailment [50]

<pre> rdf:type rdfs:domain rdfs:Resource . rdfs:domain rdfs:domain rdf:Property . rdfs:range rdfs:domain rdf:Property . rdfs:subPropertyOf rdfs:domain rdf:Property . rdfs:subClassOf rdfs:domain rdfs:Class . rdf:subject rdfs:domain rdf:Statement . rdf:predicate rdfs:domain rdf:Statement . rdf:object rdfs:domain rdf:Statement . rdfs:member rdfs:domain rdfs:Resource . rdf:first rdfs:domain rdf:List . rdf:rest rdfs:domain rdf:List . rdfs:seeAlso rdfs:domain rdfs:Resource . rdfs:isDefinedBy rdfs:domain rdfs:Resource . rdfs:comment rdfs:domain rdfs:Resource . rdfs:label rdfs:domain rdfs:Resource . rdf:value rdfs:domain rdfs:Resource . rdf:type rdfs:range rdfs:Class . rdfs:domain rdfs:range rdfs:Class . rdfs:range rdfs:range rdfs:Class . rdfs:subPropertyOf rdfs:range rdf:Property . rdfs:subClassOf rdfs:range rdfs:Class . rdf:subject rdfs:range rdfs:Resource . rdf:predicate rdfs:range rdfs:Resource . rdf:object rdfs:range rdfs:Resource . rdfs:member rdfs:range rdfs:Resource . rdf:first rdfs:range rdfs:Resource . rdf:rest rdfs:range rdf:List . rdfs:seeAlso rdfs:range rdfs:Resource . rdfs:isDefinedBy rdfs:range rdfs:Resource . rdfs:comment rdfs:range rdfs:Literal . rdfs:label rdfs:range rdfs:Literal . rdf:value rdfs:range rdfs:Resource . rdf:Alt rdfs:subClassOf rdfs:Container . rdf:Bag rdfs:subClassOf rdfs:Container . rdf:Seq rdfs:subClassOf rdfs:Container . rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property . rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso . rdf:XMLLiteral rdf:type rdfs:Datatype . rdf:XMLLiteral rdfs:subClassOf rdfs:Literal . rdfs:Datatype rdfs:subClassOf rdfs:Class . rdf:_1 rdf:type rdfs:ContainerMembershipProperty . rdf:_1 rdfs:domain rdfs:Resource . rdf:_1 rdfs:range rdfs:Resource . rdf:_2 rdf:type rdfs:ContainerMembershipProperty . rdf:_2 rdfs:domain rdfs:Resource . rdf:_2 rdfs:range rdfs:Resource </pre>

Table 5.2: RDFS axiomatic Triples [30]

$\text{rdfs:ContainerMembershipProperty, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy, rdfs:label } \cup V_{RDF}$

From this starting point we extend the vocabulary with various constructs from OWL as following:

Definition 5.2.3 (L2 Vocabulary) $V_{L2} = \{ owl:sameAs, owl:SymetricProperty, owl:TransitiveProperty, owl:inverseOf, owl:equivalentClass, owl:equivalentProperty \} \cup V_{RDFS}$

The associated set of entailment rules (see Figures 5.1 and 5.2) is then accordingly defined as a subset of the RDFS entailment rules (omitting rules **lg** and **gl**) and several additional rules covering the OWL primitives as following:

Definition 5.2.4 $Rules_{L2} = \{ rdfs2, rdfs3, rdfs4a, rdfs4b, rdfs5, rdfs6, rdfs7a, rdfs8, rdfs9, rdfs10, rdfs11, rdfs12, rdfs13 \} \cup \{ rdfp3, rdfp4, rdfp5a, rdfp6, rdfp7, rdfp8, rdfp9, rdfp10, rdfp11, rdfp12a, rdfp12b, rdfp12c, rdfp13a, rdfp13b, rdfp13c \}$

The semantics defined for via the above entailment rules are slightly weaker than their OWL counterparts, mostly for performance reasons. Following we point out the main characteristics of the chosen rule set.

- For performance reasons *L2* has only “if-conditions” for e.g. `rdf:range`, `rdf:domain`, `rdf:subClassOf`, `rdf:subPropertyOf`, `owl:TransitiveProperty`, ... instead of the stronger extensional “iff-conditions” in OWL.
- In order to still capture the intended semantics of class and property hierarchies, including reflexivity and transitivity, rules are included to make this notion explicitly visible.
- Class equivalence is cleanly layered on top of RDFS in the sense that two classes are considered equivalent if and only if they are both a subclass of each other, whereas in OWL only their extensions have to be equal. The same reasoning applies for property equivalence. This style of modelling the semantics of equivalence is rooted in the fact that equivalence i.e. between classes can already be indirectly be expressed in RDFS in this way, and that only the vocabulary to make this explicit was not available.
- Furthermore OWL treats `owl:sameAs` strictly as equivalence whereas *L2* slightly weakens its interpretation and only treats it as an equivalence relation. In order to recapture a set of essential inferences, which are unproblematic from a performance perspective, several additional rules are added.

As noted there are two different kinds of treatment for domain and range restrictions (`rdfs:range` and `rdfs:domain`). Instead of using them to infer new triples, it is also possible to treat these constructs as *constraints* on the domain and respectively range of predicates and so requires to check type information of resources.

The relevant new rules that express the intended meaning for `rdfs:domain` and `rdfs:range` constructs are defined as following, according to the definitions given in Appendix B:

Definition 5.2.5 (Domain and range consistency checking rules)

rdfs2-constr		IF	?p rdfs:domain ?t . ?s ?p ?o .
		CHECK	?s rdf:type ?t .
rdfs3-constr		IF	?p rdfs:range ?t . ?s ?p ?o .
		CHECK	?o rdf:type ?t .

6 CONCLUSIONS AND FUTURE WORK

We described a recommended knowledge representation formalism, named *L2*, that meets the requirements of LarKC. More specifically, we collected and analyzed requirements stemming from the use cases in the project and surveyed the landscape of existing knowledge representation languages on the Semantic Web and related fields.

We then examined some of the major existing inference systems for the Semantic Web for which the ability to perform data-oriented reasoning and inference over large scale data sets (in particular in industrial application). Based on the empirical analysis of the modeling primitives that are typically implemented in such systems, we identified a set of modeling primitives that (i) are implementable in a scalable way and (ii) are useful and required in industrial applications. The identified modeling primitives are considered as relevant to the LarKC project and the LarKC knowledge representation languages should be able to cover them too.

Future work includes the elaboration and refinement of the KR formalism to ensure the suitability of the formalism for typical LarKC application scenarios (as represented by the specific use case scenarios). This includes checking the adequacy of the formalism in regard of both, required expressiveness and scalability. The underlying data model will be further refined. In particular, we will investigate and define how to integrate important plug-in specific meta-information about the data set, such as information of the uncertainty or confidence in statements in the data set, or provenance-related information.

The definition of the formal semantics of *L2* by restricted entailment rules is in fact only one possible approach and should not necessarily be taken as direct algorithmic evaluation procedure, that has to and can exclusively be implemented on rule engines. We rather understand it as a principled approach for the definition of the minimal, useful and implementable KR formalism that can serve as a baseline for KR languages in LarKC. Besides this, we envision in the next version of the deliverable a closer alignment with OWL 2 and its lightweight language profiles.

A APPENDIX I

A.1 Datamodel

In this section we define the data model upon which the minimal KR formalism in LarKC is layered. RDF, as defined in [38], deliberately has a simple data model as a design goal. The data model underlying any expression in RDF is a collection of triples of the form `<subject, predicate, object>`, which form a node-and-arc-labeled directed graph.

A simple RDF graph consists of triples each to represent asserted fact or statement to be considered as true. The triple is composed by:

- Subject is the resource, which is being described;
- Predicate is a resource, which determines the type of the relationship;
- Object is a resource or a literal, which represents the “value” of the attribute.

Each triple basically denotes a statement that is asserted to be true and implies a relationship (the predicate) to hold between two things (the subject and the object). Asserting the complete graph formed by a set of triples consequently means to assert all the triples in it or in other words the intended meaning of an RDF graph amounts to the logical conjunction of all the statements denoted by the triples in the graph.

Furthermore, RDF also provides a mechanism to form “meta-statements” about RDF triples, called reification. RDF has a special *reification vocabulary* for this purpose. This vocabulary consists of the type `rdf:Statement` and the three properties `rdf:subject`, `rdf:predicate`, and `rdf:object`. Thus RDF reification can be used to represent an RDF triple as a resource in RDF. This allows other triples to describe properties of the triple and attach additional qualitative information.

For example the reification of the triple `<ex:a, ex:b, ex:c>` becomes:

```
_:x rdf:type rdf:Statement
_:x rdf:subject ex:a
_:x rdf:predicate ex:b
_:x rdf:object ex:c
```

In principle, reification allows to associate meta-information or context information with the most basic statements in a uniform manner. The semantic interpretation of reification requires the reified triple to be some concrete instance of a triple and not just an abstract grammatical form, as different concrete triples with the same triple structure could potentially exist in multiple documents. For that reason the interpretation of a reified triple is essentially a two stage process: First it is necessary to interpret the reified triple to refer to another triple, the second has to actually interpret this triple.

As can be seen in this example, at least three additional triples are needed to allow statements about the original triple. Thus, one obvious disadvantage of this approach is the so called “triple bloat” caused by using the reification vocabulary. Furthermore, querying reified statements is difficult using common query languages. For example, using SPARQL [46] requires several triple patterns in a query per single reified statement. Another issue of using reification is that it is only possible to attach

meta-information on individual triples and that it is not possible to refer to a whole graph, even though RDF will mostly be provided as a graph consisting of a set of triples. So statements need to be made per triple, and reification introduces a lot of redundant information due to that, which increases the size of the graph even more. Additionally all the meta-statements attached via reification for to all the triples in a graph might need to be kept consistent.

However, a method to express provenance, meta-data or versioning *efficiently* is crucial within LarKC. A possible extension of the RDF data model with a fourth element is suggested in [41] as a more pragmatic approach to efficiently link context to statements. The quadruple are commonly referred as quads and they are represented in the form of: $\langle S, P, O, C \rangle$. The semantics of the fourth element C named "context" is not strictly defined and has different meaning in the various quadruple stores. The authors suggest different applications of the fourth element: data source, statement or model identifier. Named RDF graph data models defined in [18], impose stricter semantics over the fourth element by defining it as RDF graph name, which in turn cannot be a blank node. The named RDF graphs are designed to be disjoint from each other and define that the blank-nodes are not shared across different graphs.

SPARQL [46] formally the notion of RDF dataset as:

Definition A.1.1 (RDF Dataset) $G, (\langle U_1 \rangle, G_1), (\langle U_2 \rangle, G_2), \dots, (\langle U_n \rangle, G_n)$

Where G and each G_n are RDF graphs, and each $\langle U_n \rangle$ is a distinct IRI. The pairs $(\langle U_i \rangle, G_i)$ are called named graphs, where $\langle U_i \rangle$ is the name of graph G_i . G is called default graph and it contains all triples, which belong to the dataset, but not to any specific named graph. A motivation to introduce default graph is the need to distinguish between data (asserted in the named graphs) and graph meta-data like provenance (asserted in the default graph).

The proposed data model in the SPARQL specification addresses the discussed conceptual drawbacks from the original RDF specification. Still, in a real life applications we can argue that the supported model may be inefficient and does not provide the required level of flexibility in cases of frequent updates or introduction of new contexts (e.g., which are anticipated for selection plug-ins for instance).

ORDI Second Generation (SG)¹ specifies a richer RDF data model which can be regarded as backward-compatible to named graphs and the simple triple model. It provides a new primitive named tripleset, which denotes a logical group to associate meta-data on a per-statement level. Thus, triplesets are defined as:

Definition A.1.2 (RDF Tripleset) $\langle S, P, O, NG, TS_1, \dots, TS_n \rangle$

where NG is an IRI to contextualize the statement (i.e. named graph). It has not formal semantics but can be used too specify the provenance of the data. $\{ TS_1, \dots, TS_n \}$ is a set of IRIs to associate the statement to additional logical groups called triplesets, which can be regarded as *view* on a set of triples. Thus, each statement could be member of multiple triplesets. From this point we will refer the defined data model as *tripleset data model*. The following formal operations are defined by the ORDI SG specification:

- Adding a new statement. The operation adds a new statement to the model.

¹<http://www.ontotext.com/ordi/index.html>

- Removal of a statement. This operation retracts a fact from the model including all associations to triplesets. Compared to the named graph model it has well defined semantics, where if two triples in different named graphs exist only one will be retracted.
- Assign a statement to a triplset. The operation associates a triple from a named graph to a triplset. If the statement does not exist the statement is not asserted.
- Un-assign a statement from a triplset. This operation removes the association statement from a triplset, however it leaves the statement in the named graph. If the statement does not exist or it is not associated with the specified triplset, the operation causes no changes to the model.
- Retrieval of statements. This operation retrieves a set of statements and their triplesets based on simple pattern match.

B APPENDIX II

B.1 Basic Definitions

RDF Terms. Let U denote the set of *URI references*, B denote the (infinite) set of *blank nodes*, and L denote the set of *literals*, i.e. data values such as strings, booleans, or XML documents. L is partitioned into the set L_p of *plain literals* and the set L_t of *typed literals*. A *typed literal* l consists of a lexical form s and a datatype URI t ; l can then be denoted as the pair $l = (s, t)$. The sets U , B , L_p , and L_t are pairwise disjoint. A *vocabulary* is a subset of $U \cup L$. Any symbol t in $U \cup B \cup L$ is called a *RDF term* and the set of RDF terms is denoted by T .

Generalized RDF Graphs. A *generalized RDF graph* G is a subset of the set

$$(U \cup B) \times (U \cup B) \times (U \cup B \cup L)$$

The elements (s, p, o) of a generalized RDF graph are called *generalized RDF triples*, which consist of a subject s , a predicate (or property) p , and an object o , respectively. We write triples as $\mathbf{s p o}$. RDF graphs [30, 38] require properties to be URI references; generalized RDF graphs, which also allow properties to be blank nodes, were introduced in [51] to solve the problem that the standard set of entailment rules for RDFS[30] is incomplete. If the projection mappings on the three factor sets of the product set given in above are denoted by π_i , the set of *RDF terms of a generalized RDF graph* G is

$$T(G) = \pi_1(G) \cup \pi_2(G) \cup \pi_3(G)$$

The set $bl(G)$ of *blank nodes of a generalized RDF graph* G is defined by $bl(G) = T(G) \cap B$. The *vocabulary of a generalized RDF graph* G is defined by $V(G) = T(G) \cap (U \times L)$.

Two generalized RDF graphs G and G' are *equivalent* if there exists a bijection $f : T(G) \leftarrow T(G')$ between the terms of the graphs such that

- $f(bl(G)) \subseteq bl(G')$, and
- $f(v) = v$ for each $v \in V(G)$, and
- $f(s) f(p) f(o) \in G'$ iff. $\mathbf{s p o} \in G$

A generalized RDF graph is *ground* if it has no blank nodes, i.e. $bl(G) = \emptyset$. Given a partial function $h : B \rightarrow T$, an *instance of a generalized RDF graph* G is the generalized RDF graph G_h obtained from G by replacing the blank nodes b in G and the domain of h by $h(b)$. Given a set S of generalized RDF graphs, a *merge* of S is a generalized RDF graph that is obtained by replacing the generalized graphs G in S with equivalent generalized graphs G' that do not share blank nodes and by taking the union of these generalized graphs G' . It is easy to see that the merge of a set of generalized RDF graphs S is uniquely defined up to equivalence (i.e. renaming of blank nodes). A merge of S will be denoted by $M(S)$.

Simple Interpretations. A *simple interpretation* [30] I of a vocabulary V is a 6-tuple

$I = (R_I, P_I, E_I, S_I, L_I, LV_I)$, where R_I is a nonempty set, called the set of *resources*, P_I is the set of *properties*, LV_I is the set of *literal values*, which is a subset of R_I that contains at least all plain literals in V , and where E_I , S_I and L_I are functions:

- $E_I : P_I \rightarrow 2^{R_I \times R_I}$
- $S_I : (V \cap U) \rightarrow (R_I \cup P_I)$
- $L_I : (V \cap L_t) \rightarrow R_I$

Here 2^X denotes the power set of the set X , i.e. the set of all subsets of X . If I is a simple interpretation of a vocabulary V , then I also denotes a function with domain V , in the following way. For $l \in L_p \cap V$, we have $I(l) = l \in LV_I$. For $l \in L_t \cap V$, $I(l) = L_I(l)$. For $a \in U \cap V$, $I(a) = S_I(a)$. If $E = \mathbf{s p o .}$ is a ground triple, then a simple interpretation I of a vocabulary V is said to *satisfy* E if $s, p, o \in V$, $I(p) \in P_I$ and $(I(s), I(o)) \in E_I(I(p))$. If G is a ground RDF graph, then I satisfies G if I satisfies each triple $E \in G$. Given a simple interpretation I and a partial function $A : B \rightarrow R_I$, a function I_A is defined that extends I by using A to give an interpretation of blank nodes in the domain of A . If $A(v)$ is defined for $v \in B$, then $I_A(v) = A(v)$. If G is any generalized RDF graph, then I satisfies G if I_A satisfies G for some function $A : bl(G) \rightarrow R_I$, i.e. if, for each triple $\mathbf{s p o .} \in G$, we have $I_A(p) \in P_I$ and $(I_A(s), I_A(o)) \in E_I(I_A(p))$. If I is a simple interpretation and S a set of generalized RDF graphs, then I satisfies S if I satisfies G for each G in S ; it is not difficult to see that I satisfies S if and only if I satisfies $M(S)$.

B.2 Entailment and Consistency Checking Rules

A rule is considered as a pair of generalized RDF graphs where variable can occur as predicate, subject, object in triples. In other words, a rule consists of two sets of triple patterns¹.

For any rule $\rho = (\rho_l, \rho_r)$, we call ρ_l the body (or left-hand side) of the rule ρ and ρ_r the head (or right-hand side) of ρ . As it is common practice for rule formalisms in deductive databases, we require that any variable occurring in the rule head ρ_h , also occurs in the body of the rule. A rule satisfying this condition is usually called *safe*. In the following, we require entailment rules to be safe rules. We also require that the body of an entailment rule cannot contain blank nodes. Both conditions are important if one wants to ensure tractability of profiles defined by entailment rules. Note, however, that for an application of a rule, as described later on, a variable occurring in the body of a rule can still be bound to a blank node in an RDF graph.

We distinguish two classes of rules: *entailment rules* and *consistency checking rules*. Entailment rules can be used to characterize expected inferences over a domain vocabulary, whereas consistency checking rules can be used to model database-style checking for the consistency of a given data set.

¹In the sense defined by the RDF Data Access Group, W3C, <http://www.w3.org/2001/sw/DataAccess/>

A rule ρ with a non-empty body and a non-empty head is called *proper rule*. Proper entailment rules will be denoted informally as

$$\text{IF } \rho_l \text{ THEN } \rho_r.$$

Informally speaking, a proper entailment rule describes under which conditions ρ_l , the statements ρ_r must hold. In this case, we can infer the statements ρ_r , whenever we detect the situation specified by ρ_l .

If only the body of an entailment rule ρ is empty, then the rule is considered as describing a certain fact or axiom. In this case, ρ is called *axiom rule* and denoted as

$$\text{AXIOMS } \rho_r.$$

If only the head of an entailment rule ρ is empty, then the rule is considered as describing a constraint, i.e. it specifies that a certain pattern of triples must be considered as being inconsistent, when occurring in some RDF triple set. In this case, ρ is called *inconsistency rule* (or constraint) and denoted as

$$\text{NOT } \rho_l.$$

Proper consistency checking rules will be denoted informally as

$$\text{IF } \rho_l \text{ CHECK } \rho_r.$$

Informally speaking, a proper consistency checking rule describes under which conditions ρ_l over a given data set G , it must be checked that G indeed satisfies the conditions ρ_r as well.

If only the body of a consistency checking rule ρ is empty, then the rule is simply written as

$$\text{CHECK } \rho_r.$$

Consistency checking rules ρ with an empty head ρ_r would not enforce any check over a given data set and therefore are not meaningful. We therefore do not allow consistency checking rules ρ with an empty head.

In the following we use N-Triples syntax for RDF to represent rule bodies and heads. Variable names start with a question mark, e.g. `?a` or `?predicate`.

Example B.2.1 ([27]) *The following proper rule captures an inference over domain specific vocabulary $V = \{ \text{ex:hasParent}, \text{ex:hasBrother}, \text{ex:hasUncle} \}$ which e.g. could be meaningful in the context of a particular language profile that is concerned with family relations between individuals:*

$$\begin{array}{ll} \text{IF} & ?a \text{ ex:hasParent } ?b . \\ & ?b \text{ ex:hasBrother } ?c . \\ \text{THEN} & ?a \text{ ex:hasUncle } ?c . \end{array}$$

Example B.2.2 ([27]) *Rules can be used for capturing the semantics of meta-modeling and to extend OWL. OWL's owl:disjointWith primitive applies to classes only. There is no similar notion for properties. Using entailment rules, we can add a primitive prof1:propertyDisjointWith to a profile (extending the RDFS profile): informally a triple p1 prof1:propertyDisjointWith p2. states that the properties p1 and p2 are disjoint, i.e. there can not be a pair of resources s and p which are inter-related via both properties p1 and p2. Further, we know that for any such triple p1 and p2 both represent properties. The intended meaning can be captured by the following inconsistency rule and axiom rules:*

```

NOT   ?p1 prof1:propertyDisjointWith ?p2 .
      ?s ?p1 ?o .
      ?s ?p2 ?o .

AXIOMS prof1:propertyDisjointWith rdfs:domain rdf:Property .
       prof1:propertyDisjointWith rdfs:range  rdf:Property .
  
```

Example B.2.3 ([27]) *We give an example of a rule, which introduces blank nodes: OWL provides the primitive owl:someValuesFrom whose semantics can be fully captured by the following two entailment rules:*

```

IF     ?c1 owl:someValuesFrom ?c2 .
      ?c1 owl:onProperty ?p .
      ?s rdf:type ?v .
THEN  ?s ?p _:blankNode .
      _:blankNode rdf:type ?c2 .

IF     ?c1 owl:someValuesFrom ?c2 .
      ?c1 owl:onProperty ?p .
      ?s ?p ?v .
      ?v rdf:type ?c2 .
THEN  ?s rdf:type ?c1 .
  
```

Example B.2.4 *The semantics of domain and range restrictions in RDFS is not as uniquely defined as one would expect. It is mentioned in [8, Section 4]:*

The RDF Vocabulary Description language provides a mechanism for describing this information, but does not say whether or how an application should use it. For example, while an RDF vocabulary can assert that an author property is used to indicate resources that are instances of the class Person, it does not say whether or how an application should act in processing that range information. Different applications will use this information in different ways. For example, data checking tools might use this to help discover errors in some data set, an interactive editor might suggest appropriate values, and a reasoning application might use it to infer additional information from instance data.

In particular, in [51, 50, 30] the semantics of domain and range restrictions is considered only under the last perspective, i.e. used to infer the additional type information for resources in the data set.

Using consistency checking rules we can capture an alternate semantics of domain and range restrictions in RDFS (i.e. the vocabulary `rdfs:domain` and `rdfs:range`) that embodies the first mentioned perspective on domain and range restrictions in the quote above. For this purpose we can use the the following consistency checking rules:

rdfs2-constr	IF ?p rdfs:domain ?t . ?s ?p ?o . CHECK ?s rdf:type ?t .
rdfs3-constr	IF ?p rdfs:range ?t . ?s ?p ?o . CHECK ?o rdf:type ?t .

We give a formal definition of entailment and consistency checking rules on (generalized) RDF graphs in the following. In our definition of rules we use a set of rule variables X which is assumed to be disjoint from the set T of RDF terms, i.e. $X \cap T = \emptyset$. Rule variables will also briefly be called variables.

Rule Graph. A rule graph G is defined to be a set of triple patterns, i.e. a subset of the product set

$$(U \cup B \cup X) \times (U \cup B \cup X) \times (U \cup B \cup L \cup X)$$

Given a rule graph G , we denote the union of the projection mappings π_i on the three factor sets of the product set given above, applied to G , by

$$\pi(G) = \pi_1(G) \cup \pi_2(G) \cup \pi_3(G)$$

The set of *variables of a rule graph* G is denoted by $var(G) = \pi(G) \cap X$, the set of *blank nodes of* G by $bl(G) = \pi(G) \cap B$, and the *vocabulary of* G by $V(G) = \pi(G) \cap (U \cup L)$.

Two kinds of *instances of rule graphs* are defined: one with respect to variables and one with respect to blank nodes. Given a rule graph G and a function $\phi : var(G) \rightarrow T$, the *instance of* G *with respect to the variable instantiation* ϕ is the generalized RDF graph G_ϕ obtained from G by replacing the variables $v \in var(G)$ by $\phi(v)$. Similarly, given a rule graph G and a partial function $h : B \rightarrow T$, the *instance of* G *with respect to the blank node instantiation* h is the rule graph G_h obtained from G by replacing the blank nodes v in G and the domain of h by $h(v)$. Given a rule graph G combined with $h : B \rightarrow T$ and $\phi : var(X) \rightarrow T$, $G_{h\phi}$ is the instance of G_h with respect to ϕ .

Entailment and Consistency Checking Rules. An *entailment rule* is defined as a pair of rule graphs $\rho = (\rho_l, \rho_r)$ such that

- at least one of rule graphs ρ_l, ρ_r is non-empty, and
- $var(\rho_r) \subseteq var(\rho_l)$, and
- $bl(\rho_l) = \emptyset$

An *consistency checking rule* is defined as a pair of rule graphs $\rho = (\rho_l, \rho_r)$ such that

- at least the rule graph ρ_r is non-empty, and
- $bl(\rho_l) = \emptyset$

If $\rho = (\rho_l, \rho_r)$ is a rule, then ρ_l is called its left-hand side or body, and ρ_r is called its right-hand side or head. Given a rule ρ , the set of *variables of ρ* is denoted by $var(\rho) = var(\rho_l)$, the set of *blank nodes of ρ* by $bl(\rho) = bl(\rho_r)$, and the *vocabulary of ρ* by $V(\rho) = V(\rho_l) \cup V(\rho_r)$.

If R is a set of rules, then $V(R) = \bigcup_{\rho \in R} V(\rho)$. An entailment rule ρ is said to *introduce blank nodes* if $bl(\rho) \neq \emptyset$. A rule ρ is called *finite* if the rule head ρ_r and the rule body ρ_l are both finite. A rule ρ is called a *proper rule* if the rule head ρ_r and the rule body ρ_l are both nonempty. An entailment rule is called an *axiom rule* if $\rho_l = \emptyset$, and an *inconsistency rule* (or *constraint*) if $\rho_r = \emptyset$.

B.3 Semantics for Entailment and Consistency Checking Rules

We can now define the meaning of an entailment and consistency checking rules and their effects in a model-theoretic sense following [50]; this is achieved in two steps: first we define when a rule is satisfied by an interpretation, and second we define what statements (or triples) are entailed by a set of rules R , i.e. the notion of simple R -entailment.

Satisfaction of Rules. Given a simple interpretation I and a partial function $Z : X \rightarrow R_I$, a function I_Z is defined that extends I by setting $I_Z(v) = Z(v)$ if $Z(v)$ is defined for $v \in X$. If, in addition, a partial function $A : B \rightarrow R_I$ is given, a function I_{ZA} is defined that extends I_Z further by setting $I_{ZA}(v) = A(v)$ if $A(v)$ is defined for $v \in B$. If G is any rule graph, I a simple interpretation and $Z : var(G) \rightarrow R_I$ a function, then I_Z is said to *satisfy G* if there is a function $A : bl(G) \rightarrow R_I$ such that for each triple pattern $\mathbf{s p o} . \in G$ we have $I_{ZA}(p) \in P_I$ and $(I_{ZA}(s), I_{ZA}(o)) \in E_I(I_{ZA}(p))$. A simple interpretation I *satisfies a rule ρ* if $I(p) \in P_I$ for each $p \in U$ that appears in predicate position in a triple pattern in ρ_l or ρ_r , and if I also satisfies the following conditions:

- If ρ is an axiom rule, then I satisfies ρ_r
- If ρ is a proper entailment rule and $Z : var(\rho) \rightarrow R_I$ a function such that I_Z satisfies ρ_l , then I_Z also satisfies ρ_r
- If ρ is an inconsistency rule, then there is no function $Z : var(\rho) \rightarrow R_I$ such that I_Z satisfies ρ_l
- If ρ is a consistency checking rule and $Z : var(\rho) \rightarrow R_I$ a function such that I_Z satisfies ρ_l , then I_Z also satisfies ρ_r

Note, proper entailment rules and consistency checking rules are satisfied by an interpretation under the same circumstances. The meaning of both types of rules will differ later only when we consider what can be entailed from a data set given a set of rules R .

Simple R-Interpretations and Simple R-Entailment. Let R be any set of rules and let $R_{entail} \subseteq R$ denote the set of all the entailment rules in R . A *simple R-interpretation* of a vocabulary V is a simple interpretation of $V \cup V(R)$ that satisfies each rule $\rho \in R$. A set S of generalized RDF graphs is called *simple R-consistent* if there is a simple R -interpretation that satisfies S . The set of *simple R-axiomatic triples* is the generalized RDF graph that is obtained by taking the merge of the generalized RDF graphs ρ_r that occur as heads in the axiom rules of R .

Given a set of rules R and an extended RDF graph G , an *R-clash* is a generalized RDF graph that forms an instance $\rho_{l\phi}$ of the body ρ_l of an inconsistency rule $\rho \in R$ for a function $\phi : var(R) \rightarrow T$.

A set S of generalized RDF graphs *simple R-entails* a generalized RDF graph G if each simple R_{entail} -interpretation I that satisfies S also satisfies G . This situation is denoted as $S \models_R G$ as usual.

Note, that for the notion of entailment only the set of entailment rules in R is considered; in other words, consistency checking rules do not play any role when we consider what can be simple R -entailed from a data set S ; they do not change what can be entailed by an application from a data set S from a (sub)set of entailment rules R_{entail} . However, if we consider the simple R -consistency of a data set S , then the consistency checking rules in R must be taken into account too.

In the following, we discuss the syntax and semantics following [51, 50].

For a rule set R and a generalized RDF graph R , we define a generalized RDF graph which represents the deductive closure of G with respect to the rules in R :

The *simple R-closure* G_R of G with respect to R is the generalized RDF graph G' which is constructed from G as follows: first, we create a graph G_0 from G by adding all R -axiomatic triples in such a way, that G does not contain any blank node that appears in the merge of the generalized RDF graphs ρ_r that occur as heads of axiom rules $\rho \in R$. Then, we extend G_0 iteratively to extended RDF graphs G_1, G_2, \dots : For a given G_n , we construct G_{n+1} by applying entailment rules to G_n . More precisely, for each proper entailment rule $\rho \in R_{entail}$, one application is made for each instance $\rho_{l\phi}$ of the body ρ_l for a function $\phi : var(\rho) \rightarrow T(G_n)$, where $\phi(x) \in U \cup B$ for each $x \in (\pi_1(\rho_r) \cup \pi_2(\rho_r) \cap X)$, such that $\rho_{l\phi} \subseteq G_n$, and, if ρ introduces blank nodes, such that there is no function $h : bl(\rho_r) \rightarrow T(G_n)$ such that $\rho_{r\phi h} \subseteq G_n$. Finally, we set $G' = \bigcup_{n \geq 0} G_n$.

We will see below that that the restricted use of proper entailment rules that introduce blank nodes is in general sufficient to determine R -entailment.

The following properties about simple R -satisfaction and simple R -entailment can be established:

Proposition B.3.1 (Simple R-Entailment Lemma) *Let R be a set of rules, S be a set of generalized RDF graphs and G a generalized RDF graph. Let H be the simple R -closure $M(S)_R$ of $M(S)$. Then, $S \models_R G$ if and only if either H contains an instance of G as a subset or H contains an R -clash (for some inconsistency rule $\rho \in R$).*

This proposition gives us a syntactic criterion for checking simple R -entailment. As a corollary we can derive upper bounds on the complexity of the simple R -entailment problem:

Corollary B.3.2 *The simple R -entailment relation $S \models_R G$ between finite sets S of finite generalized RDF graphs, finite sets R of finite rules that do not introduce blank nodes, and finite generalized RDF graphs G is decidable and in PSPACE. If there is a fixed bound on the size of the rule bodies for the rules in R , then this problem is in NP and in P if the target graph G is ground.*

This means that essentially by restricting the rules in R syntactically, we can define a specific entailment notion which can be decided efficiently, if blank nodes are not used in the target graph.

Similar properties hold for checking R -consistency:

Let G be a generalized RDF graph and ρ a consistency checking rule. We say that ρ is *violated in G* , if there exists an instance $\rho_l\phi$ of the body ρ_l for a function $\phi : \text{var}(\rho) \rightarrow T$, where $\phi(x) \in U \cup B$ for each $x \in (\pi_1(\rho_r) \cup \pi_2(\rho_r) \cap X)$, such that $\rho_l\phi \subseteq G$, and, if ρ introduces blank nodes, there is a function $h : \text{bl}(\rho_r) \rightarrow T$ such that $\rho_r\phi h \not\subseteq G$. We call a pair (ϕ, h) for which the condition is satisfied a *violation witness* for ρ in G .

Proposition B.3.3 (Simple R -Consistency Lemma) *Let R be a set of rules, S be a set of generalized RDF graphs and let H be the simple R -closure $M(S)_R$ of $M(S)$. Then, S is R -consistent if and only if H does not contain an R -clash (for some inconsistency rule $\rho \in R$) and there is no consistency checking rule $\rho \in R$ that is violated in H .*

Corollary B.3.4 *The problem to determine if a finite set S of finite generalized RDF graphs is simple R -consistent for a finite set R of finite rules that do not introduce blank nodes is decidable and in PSPACE. If there is a fixed bound on the size of the rule bodies for the rules in R , then this problem is in P.*

We do not give detailed formal proofs in this document but leave them as future work. The propositions can be proven using the same techniques that have been used in [51, 50] with some minor modifications and extensions.

REFERENCES

- [1] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyashev. The DL-Lite family and relations. Technical report, Technical Report BBKCS-09-03, School of Computer Science and Information Systems, Birbeck College, London, 2009. Available at <http://www.dcs.bbk.ac.uk/research/techreps/2009/bbkcs-09-03.pdf>.
- [2] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope. *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, 19:364, 2005.
- [3] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope Further. *Proceedings of the OWLED Workshop*, 2008.
- [4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P F. Patel-Schneider. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
- [6] Sean Bechhofer and Raphael Volz. Patching syntax in OWL ontologies. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, 2004.
- [7] R.J. Brachman and H.J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI-84)*, pages 34–37, 1984.
- [8] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*, 2, 2004.
- [9] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C, 2004.
- [10] Jos de Bruijn. Wsml abstract syntax and semantics. Working Draft D16.3 v0.3, WSMML, 2007.
- [11] Jos de Bruijn, Thomas Eiter, Axel Polleres, and Hans Tompits. On representational issues about combinations of classical theories with nonmonotonic rules. In *Proceedings of the 1st International Conference on Knowledge Science, Engineering and Management (KSEM2006)*, pages 1–22. Springer, 2006.
- [12] Jos de Bruijn, Thomas Eiter, Axel Polleres, and Hans Tompits. Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI2007)*, pages 304–309, Hyderabad, India, January 6–12 2007. AAAI Press.
- [13] Jos de Bruijn and Stijn Heymans. A semantic framework for language layering in WSMML. In *Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR2007)*, pages 103–117, Innsbruck, Austria, June 7–8 2007. Springer.

- [14] Jos de Bruijn and Stijn Heymans. WSML ontology semantics. Working Draft d28.3, WSML Working Group, 2007.
- [15] Jos de Bruijn and Stijn Heymans. On the relationship between description logic-based and f-logic-based ontologies. *Fundamenta Informaticae*, 2008. Accepted for publication.
- [16] Jos de Bruijn, David Pearce, Axel Polleres, and Agustín Valverde. Quantified equilibrium logic and hybrid rules. In *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems (RR2007)*, pages 58–72, Innsbruck, Austria, June 7–8 2007. Springer.
- [17] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable Description Logics for Ontologies. *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, 20(2):602, 2005.
- [18] J.J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM New York, NY, USA, 2005.
- [19] D. Connolly, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. DAML+ OIL (March 2001) Reference Description. *W3C Note*, 18, 2001.
- [20] Jos de Bruijn, Dieter Fensel, Mick Kerrigan, Uwe Keller, Holger Lausen, and James Scicluna. *Modeling Semantic Web Services: The Web Service Modeling Language*. Springer, 06 2008.
- [21] Mike Dean and Guus Schreiber. OWL web ontology language reference. Recommendation 10 February 2004, W3C, 2004.
- [22] C. Dolbear, G. Hart, and J. Goodwin. What OWL has done for geography and why we don’t need it to map read. *Workshop OWL: Experiences and Directions*, pages 10–11, 2006.
- [23] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*. AAAI Press, 2004.
- [24] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web services – The Web service Modeling Ontology*. Springer, 2006.
- [25] Allen Van Gelder, Kenneth Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
- [26] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

-
- [27] Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors. *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, volume 3729 of *Lecture Notes in Computer Science*. Springer, 2005.
- [28] B. Groszof, I. Horrocks, R. Volz, and S. Decker. *Description Logic Programs: Combining Logic Programs with Description Logic*. 2003.
- [29] Benjamin N. Groszof, Ian Horrocks, Raphael Volz, and Stefan Decker. *Description logic programs: Combining logic programs with description logic*. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary, 2003.
- [30] P. Hayes and B. McBride. *RDF Semantics. W3C Recommendation*, 10, 2004.
- [31] P. Hayes and C. Menzel. *A semantics for the knowledge interchange format. IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*, 2001.
- [32] Patrick Hayes. *RDF semantics. Technical report, W3C*, 2004. W3C Recommendation 10 February 2004.
- [33] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai Wang. *The manchester OWL syntax*. In *Proceedings of the workshop OWL: Experiences and Directions 2006*, Athens, GA, USA, 2006.
- [34] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. *Practical reasoning for very expressive description logics. Logic Journal of the IGPL*, 8(3):239–264, May 2000.
- [35] Herman J. ter Horst. *Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the owl vocabulary. Journal of Web Semantics*, 3(2–3):79–115, 2005.
- [36] U. Hustadt, B. Motik, and U. Sattler. *Reasoning for Description Logics around SHIQ in a Resolution Framework. FZI, Karlsruhe Germany*, 2004.
- [37] Michael Kifer, Georg Lausen, and James Wu. *Logical foundations of object-oriented and frame-based languages. JACM*, 42(4):741–843, 1995.
- [38] G. Klyne, J.J. Carroll, and B. McBride. *Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation*, 10, 2004.
- [39] Markus Krtzsch, Sebastian Rudolph, and Pascal Hitzler. *Elp: Tractable rules for owl 2*. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *LNCS*, pages 649–664. Springer, OCT 2008.
- [40] John W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.
- [41] R. MacGregor and I.Y. Ko. *Representing Contextualized Data using Semantic Web Tools*. In *Practical and Scalable Semantic Systems (workshop at 2nd ISWC)*, 2003.

- [42] D.L. McGuinness, F. van Harmelen, et al. OWL Web Ontology Language Overview. *W3C Recommendation*, 10:2004–03, 2004.
- [43] B. Motik and I. Horrocks. OWL 2 Web Ontology Language. *W3C Working Draft*, W3C, 2008.
- [44] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can owl and logic programming live together happily ever after? In *Proc. of the 5th Int. Semantic Web Conf. (ISWC 2006)*, Athens, GA, USA, November 5 – 9 2006.
- [45] Boris Motik and Riccardo Rosati. A faithful integration of description logics with logic programming. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India, January 6–12 2007.
- [46] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Recommendation 15 January 2008, W3C, 2008.
- [47] Teodor C. Przymusiński. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5(2):167–205, 1989.
- [48] D.A. Randell, Z. Cui, A. Cohn, B. Nebel, C. Rich, and W. Swartout. A spatial logic based on regions and connection. *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 165–176, 1992.
- [49] Riccardo Rosati. $\mathcal{DL}+log$: Tight integration of description logics and disjunctive datalog. In *KR2006*, 2006.
- [50] Herman J. ter Horst. Combining rdf and part of owl with rules: Semantics, decidability, complexity. In Gil et al. [27], pages 668–684.
- [51] Herman J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *J. Web Sem.*, 3(2-3):79–115, 2005.
- [52] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe, 2004.
- [53] WSML Working Group. WSML language reference. Working Draft D16.1 v0.3, WSML, 2008.
- [54] Z. Wu, G. Eadon, S. Das, E.I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1239–1248, 2008.